MASTER EN SCIENCES INFORMATIQUES

Réalisation d'un langage graphique pour ASAX

Gumiro, Laurien

Award date:
1999

FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX

NAMUR

INSTITUT D'INFORMATIQUE

Année académique 1998-1999

REALISATION D'UN LANGAGE GRAPHIQUE POUR ASAX

Laurien GUMIRO

Mémoire présenté en vue de l'obtention du grade de Licencié en Informatique

RESUME

Ce travail a pour but la réalisation et l'implémentation d'un interface graphique pour ASAX. Partant d'un interface donné dans le manuel de l'utilisateur ASAX, une démarche de conception basé sur l'analyse de la tâche a été effectuée en vue d'apporter d'éventuelles améliorations à l'interface graphique du manuel. La méthode utilisée n'apporte pas de modifications significatives.

Le langage utilisé pour programmer l'interface graphique est Tcl/Tk, un langage à script facile à apprendre et pour lequel il existe actuellement des générateurs du code C pour les problèmes de portabilité et de lisibilité du code. Toutes les fonctionnalités relatives à l'exécution de l'analyse ont été réalisées. Les travaux ultérieurs pourront se concentrer à la programmation de l'aide fournie à l'utilisateur et des boîtes de message fournies par ASAX. Il serait également intéressant d'intégrer les aspect off-line d'ASAX dans l'interface graphique tels que l'installation et la génération de l'exécutable d'ASAX.

ABSTRACT

The aim of this thesis is to build a graphical user interface for ASAX. The Task Knowledge Structure (TKS) methodology have been used during the conception in hope to improve the ASAX graphical user interface (GUI) described in User's Guide manual. The TKS methodology haven't made important changes comparing with the previous GUI proposed in the manual.

The simple and powerful scripting language named Tcl/Tk have been used for programming. Most functions important for analysis have been implemented in this work. The help functions and some other functions have to be realised during future work.

REMERCIMENTS

Je voudrais remercier particulièrement Monsieur le Professeur Bedouin Le Charlier, promoteur de ce travail, pour ses conseils et qui m'a suivi tout au long de ce travail. Je lui témoigne toute ma reconnaissance.

Je tiens également à remercier Monsieur Abdelaziz Mounji pour sa disponibilité et qui m'a fait partagé ses connaissances d'ASAX.

Je remercie Monsieur Efrem Mbaki pour ses conseils dans la conception de l' interface graphique.

Je remercie Majid pour son aide concernant la documentation pour ASAX.

Je dédie ce travail à toute ma famille ainsi qu'à tous mes amis pour leur soutien.

Enfin j'adresse mes remerciements à tous ceux qui m'ont aidé d'une façon quelconque dans ce travail.

TABLE DES MATIERES

INTRODUCTION

Ce travail s'inscrit dans le cadre du projet ASAX (Advanced Security Audit-Trails Analysis on UNIX). L'objectif est de réaliser et d'implémenter un interface graphique pour fournir à l'utilisateur un environnement de travail convivial pour l'utilisation d'ASAX.

La plate-forme supportée actuellement par ASAX est UNIX. Or, nous ne connaissions pas beaucoup de langages qui permettent de développer des interfaces graphiques sous UNIX. Nous étions intéressés par un langage de haut niveau, facile à apprendre et qui permettrait de faciliter la programmation. Notre choix s'est porté sur Tcl/Tk, un langage à script qui est utilisé dans beaucoup d'applications entre autre pour programmer des interfaces graphiques.

Nous sommes parti d'un interface graphique qui est décrit dans le manuel *User's Guide* d'ASAX et nous avons appliqué la méthode de conception des interfaces graphiques vue au cours d'IHM. Le but était de pouvoir apporter d'éventuelles améliorations à ce qui était proposé bien que nous ne connaissions pas la méthode qui a été utilisée dans ce manuel. Nous verrons que la démarche poursuivie n'apporte pas de changements importants par rapport à la conception de départ.

Dans le premier chapitre, nous commencerons par rappeler en quoi consiste ASAX, ses principales caractéristiques. Nous parlerons du format de fichiers d'audit appelés NADF qui est utilisé dans ASAX et du langage RUSSEL (Rule-baSed Sequence Evaluation Language). Nous présenterons enfin l'interface utilisateur actuellement disponible.

Le chapitre suivant est consacré au langage Tcl/Tk. L'objectif de ce chapitre est de présenter les notions de base du langage ainsi que celles que nous avons utilisé dans la réalisation du mémoire. La première partie de ce chapitre présente le langage à script Tcl. La seconde partie présente Tk qui est une extension du Tcl et qui est utilisé pour réaliser des interfaces graphiques. Il est important de connaître Tcl car les objets graphiques fournis pat Tk sont rassemblés en utilisant des scripts Tcl.

Le dernier chapitre est relatif à l'interface graphique. Nous présenterons d'abord la démarche poursuivie et les résultats obtenus. Ensuite nous ferons le point sur l'interface réalisé, la méthode utilisée et nous commenterons quelques fonctionnalités qui nous ont causé quelques difficultés dans la programmation. Nous présenterons enfin l'état d'avancement du travail et ce qui reste à faire.

CHAPITRE 1 : LE PROJET ASAX

Nous allons commencer par introduire ASAX. Après avoir exposé ses principales caractéristiques, nous parlerons des fichiers d'audit normalisés et du langage RUSSEL ; pour finir nous présenterons l'interface en mode commande fourni par ASAX.

1.1. Historique

(Réf. 6)
ASAX a été développé comme un système d'analyse d'audit trail pour BS2000 et SINIX, deux systèmes d'exploitation radicalement opposés de Siemens-Nixdorf A.G. Le design et l'implémentation d'ASAX ont été menés en parallèle avec ceux des générateurs d'audit trail de BS2000 et SINIX et donc ASAX a été développé au moment où l'on disposait de peu de connaissances sur la sémantique des enregistrements d'audit ainsi que sur leur format. Sur ce, ASAX a été conçu avec deux objectifs importants. Premièrement, la conception d'un format d'enregistrement canonique, hautement flexible dans lequel tous les enregistrements des différents systèmes d'exploitation peuvent être aisément convertis. Deuxièmement, la conception d'un langage à base de règles efficace et puissant qui permet d'exprimer et de détecter aléatoirement les patterns complexes d'enregistrements d'audit.

1.2. Les grands traits d'ASAX

(Réf. 4, 5)
ASAX est un outil d'analyse universel, puissant, efficace, portable et facilement extensible.

1. L'universalité
   ASAX est un outil universel d'analyse dans le sens où il est capable d'analyser n'importe quel fichier séquentiel. Cet objectif est atteint :
   - En transformant n'importe quel fichier séquentiel, également appelé ADF, en un format normalisé d'audit, également appelé NADF.
   - En utilisant un analyseur adapté uniquement au format NADF

   La conversion est assurée par le fait que le format NADF est simple et flexible de telle sorte que n'importe quel format d'origine peut y être facilement converti. Plus précisément, un fichier NADF est simplement une séquence d'enregistrements dans le format NADF. La partie qui transforme un fichier séquentiel en un fichier NADF est appelé un adaptateur de format.

2. La puissance
   Vu la complexité de scénarios d'attaque dans les systèmes d'exploitation, il faut pouvoir représenter et détecter toute sorte d'intrusions possibles. Ceci est atteint en utilisant un langage à base de règles spécifiquement conçu pour l'analyse des fichiers séquentiels. C'est le langage RUSSEL.
   Toutefois, dans l'analyse d'audit trail, un langage à base de règles ne peut permettre d'encoder toute sorte de connaissance déclarative ou de faire un raisonnement général sur cette connaissance. Pour atteindre ce but, on utilise un langage à base de règles de façon bien établie : reconnaître les patterns particuliers dans les fichiers séquentiels et activer les actions appropriées, par exemple activer un alarme, envoyer un message. L'idée de RUSSEL est de profiter d'une telle utilisation pour faire un langage efficace et facile à utiliser autant que possible.
   Donc, RUSSEL peut être considéré comme un langage procédural incluant en particulier les structures de contrôle prédéfinies qui sont appropriées pour faire un raisonnement sur

les séquences d'enregistrements. Cette structure de contrôle est basée sur un mécanisme de déclenchement de règles qui peut être résumé comme suite :

- un audit trail est analysé séquentiellement enregistrement par enregistrement au moyen d'un ensemble de règles. A un moment donné, il y a un enregistrement courant qui est analysé et un ensemble de règles qui sont actives.
- Les règles actives mémorisent toute connaissance concernant l'analyse passée. Cette connaissance est alors appliquée à l'enregistrement courant en exécutant des règles pour cet enregistrement. Ce processus génère à son tour de nouveaux règles qui seront appliquées ultérieurement.
- Du point de vue du programmeur, une règle est comme une procédure paramétrée qui peut contenir un type d'actions particulières : le déclenchement de règles.
- Déclencher une règle implique de fournir les paramètres effectifs de la règle et spécifier quand la règle sera déclenchée : pour un enregistrement courant, pour le prochain enregistrement ou à la fin de l'analyse.
- Le processus est initialisé par un ensembles de règles activées pour le premier enregistrement.

Donc, programmer une requête pour l'analyse d'un audit trail consiste à écrire un certain nombre de règles adéquates qui seront activées conformément au schéma de contrôle global donné ci–dessus. Le fait que les audit trails sont triés chronologiquement joue un rôle essentiel dans cette approche : l'analyse avancée d'audit trails consiste principalement à rechercher des sous séquences chronologiques complexes dans le flux lié à tous les événements enregistrés.

3. L'efficacité

   Cet objectif est critique pour la détection des intrusions car l'analyse est effectuée sur des fichier en général très gros. ASAX possède deux principaux caractéristiques permettant d'être efficace :

   - Le langage RUSSEL permet une analyse d'audit trail en une et uniquement une seule passe. Les informations nécessaires sur le passé de l'enregistrement courant sont encapsulées dans un ensemble de règles actives. En plus RUSSEL supporte un appel des routines externes, l'information sur le passé peut être mémorisée dans les structures de données du C, elle pourra être accessible via des routines externes.
   - Les étapes répétitives sont optimisées autant que possible via des techniques efficaces et sophistiquées d'implémentation. Voici quelques idées principales sur l'implémentation :
     a) les règles RUSSEL sont traduites en un code interne optimisé qui peut être exécuté efficacement par une machine abstraite appropriée.
     b) Le système évaluera dans la plupart des cas les règles qui impliquent différents champs de données dans l'enregistrement courant, est donc un accès efficace à l'enregistrement courant est critique. L'enregistrement courant a une longueur variable et un format libre. Donc, avant d'appliquer les règles actives à un enregistrement courant, l'enregistrement est d'abord analysé en vue de créer une table d'indirection optimisée contenant les pointeurs aux différentes données d'un enregistrement. Durant l'analyse de l'enregistrement courant, les données audit seront accédées via cette table d'indirection.
     c) Le système conserve à un moment donné trois ensembles de règles : celles qui sont actives pour l'enregistrement courant, celles qui seront activées pour le prochain enregistrement et celles qui seront activées à la fin de l'analyse de tout l'audit trail. Chaque ensemble est représenté par une liste chaînée où chaque cellule contient deux types d'information : un pointeur au code interne de la règle et une

liste de valeurs de paramètres effectifs. Remarquons que plusieurs instances d'une même règles peuvent exister et peuvent toutes utiliser le même code interne. Le passage de l'enregistrement courant à l'enregistrement suivant est réalisé de façon simple en se déplaçant de l'ensemble de règles actives pour l'enregistrement courant vers l'ensemble de règles actives pour l'enregistrement suivant, et en réinitalisant l'ensemble de règles actives pour l'enregistrement suivant à l'ensemble vide.

d) Quand le traitement d'une règle est commencé, les paramètres effectifs sont recopiés dans un emplacement qui est le même pour toutes les règles. Ceci permet un accès direct aux valeurs des paramètres effectifs.

4. La portabilité
   ASAX est aisément portable sur n'importe quelle machine et sur n'importe quel système d'exploitation.

5. L'extensibilité
   Le langage RUSSEL supporte des fonctions très variées notamment celles les fonctionnalités de reporting, statistiques, d'alarme. En plus RUSSEL a été conçu de façon à avoir une interface avec le langage C. On peut donc aisément étendre facilement les fonctionnalités d'ASAX.

1.3. Le format NADF

(Réf. 5)
L'objectif d'ASAX est de fournir un outil universel d'analyse d'audit trails compatible avec un large spectre de systèmes d'exploitation. La raison d'être de ce raisonnement est double :

1) pouvoir analyser n'importe quel audit trail, ce dernier doit être préalablement converti en un format approprié.

2) assurer un maximum d'efficacité qui reste le plus grand objectif dans l'analyse d'audit trails.

Pour cela, on a alors défini un format d'audit trail normalisé lequel est suffisamment flexible de telle sorte que n'importe quel audit trail existant peut y être converti de façon très simple. L'analyse d'audit trails est donc effectuée uniquement sur les audit trails normalisés. Un adaptateur de format pourrait être fourni pour convertir n'importe quel audit trail native en un format NADF. L'adaptateur de format a été réalisé pour les audit trails qui existent pour BS2000, SINIX, SunOS 4.1.1, et SunOS 5.x.
La conversion est simple à cause du format NADF qui est très simple et flexible. Un fichier NADF est une séquence chronologique d'enregistrement, où chaque enregistrement consiste en une liste de données audit. Chaque donnée audit est composée d'un identifiant de la donnée et de sa valeur.
L'adaptateur de format doit en plus générer d'autres fichiers auxiliaires qui conservent la correspondance entre les formats externes et internes des données audit : les tables d'association, les routines de décodage, etc. Ceci permet d'exprimer les requêtes d'analyse en utilisant un format externe.

Représentation physique des fichiers NADF

Dans un fichier audit, chaque champ d'un enregistrement est constitué du couple (fname,value) où *fname* est le nom du champ et *value* sa valeur.
Pour une représentation physique exacte, on associe à chaque champ un nombre naturel qu'on appelle *audit data identifier* qui identifie de façon non-ambiguë chaque champ.
Représentation d'une donnée audit :
Soit un champ (*fname,value*), sa représentation est constituée d'éléments suivants :
Identifiant : représentation binaire d'un entier court non signé de l'identifiant d'une donnée audit associé à chaque nom de champ *fname* ;
Longueur : représentation binaire d'un entier non signé de la longueur de la valeur (*value*) du champ en bytes ;
Value : chaîne de bytes représentant la valeur *value* elle-même.
Représentation d'un enregistrement :
La représentation d'un record est constituée des éléments suivants :
Longueur total d'un record : représentation binaire à quatre bits de la longueur en bytes de tout l'enregistrement NADF.
Séquence de champs : séquence contiguë de la représentation de chaque champ dans un enregistrement. La séquence est triée par ordre ascendant des identifiants des données audit.
Exemple :

```
Struct {
        Char directory[10] ;
        Int uid ;
        Char filename[12] ;
} Record = {« /tmp », 123, « /etc/passwd »} ;
```

Supposons que les champs des données audit *directory, uid,* et *filename* ont pour identifiants 4, 1, et 2 respectivement. La représentation NADF résultante est représentée ci-dessous.

Figure 1.1 Un enregistrement NADF (réf. 6)

1.4. Le langage RUSSEL

(Réf. 6)
Le concept de module qui est utilisé dans le langage correspond à un fichier source contenant une collection de règles et les déclarations de variables globales. Quelques unes de ces variables et règles peuvent être déclarés internes (dans ce cas, elles sont visibles aux règles qui sont déclarées dans le même fichier source) ou externe (dans ce cas, elles sont visibles à tous les modules). La déclaration de variable global interne (respectivement externe) ou d'une règle est précédée par le mot clé **internal** (respectivement **external**). Un grand fichier source qui implémente une analyse complexe peut par conséquent être divisé en plusieurs fichiers sources et chaque fichier est un module. Chaque module implémente une analyse particulière et interface les autres modules à travers les déclarations de variables globales externes et/ou de règles externes. Chaque module peut déclarer optionnellement ses propres *actions d'initialisation (init action)* qui sont par définition internes. Dans certains cas, il convient de rassembler toutes les variables qui sont partagées par toutes les modules en un module. Ce dernier a sa propre *init action* qui initialise ces variables partagées.

Le langage fournit aussi des mécanismes qui permettent à un module d'utiliser un autre module. Ce dernier peut aussi utiliser un autre module et ainsi de suite.
La déclaration d'une règle comporte le nom de la règle, la déclaration d'une liste de paramètres formels, la déclaration d'une liste de variables, et une partie action qui est le corps principal de la règle. Un module doit déclarer une variable globale externe si une règle quelconque référence cette variable ; un module n'a pas besoin de déclarer une règle externe s'il n'est référencée par aucune de ses règles.
L'exécution d'un programme RUSSEL effectue l'analyse séquentielle d'un audit trail, un enregistrement à chaque instant. L'audit trail qui est analysé à un moment donné est appelé *current audit record*. Ce dernier peut être référencé dans le corps de la règle par les noms de ses champs tels que USER_ID ou TIMESTAMP. Donc, les noms des champs sont réellement traités comme les variables globales déclarées implicitement dans le programme RUSSEL. Il n'y a aucune notion de types d'enregistrement en RUSSEL. Donc, n'importe quel champ possible peut être présent ou absent dans l'enregistrement courant en fonction de la nature de l'événement représenté par l'enregistrement. En considérant l'ensemble de tous le noms des champs qui peuvent être présent dans l'enregistrement d'un audit, lire le prochain d'audit a pour effet d'affecter une valeur aux champs qui sont présents tout en laissant les autres qui ne sont pas présents. Il est possible de tester si un champ est présent ou non.
Une instance de la déclaration d'une règle (aussi appelé règle active) est composée de la déclaration de la règle et une liste de valeurs de paramètres réels. Chaque instance d'une règle est un membre de trois ensembles d'instances de règles :

1) *Current* : l'ensemble d'instances de règles qui ont été activées pour l'enregistrement courant mais pas encore exécutées.

2) *Next* : l'ensemble d'instances de règles qui sont activées pour l'analyse de l'enregistrement suivant.

3) *Completion* : l'ensemble d'instances de règles qui ont été activées pour être exécutées à la fin de l'analyse de tout le fichier audit. Le traitement de l'enregistrement courant consiste en l'exécution de toutes les instances de règles dans l'ensemble *current*, une à la fois. L'instance de la règle qui est exécutée pour un enregistrement courant est appelée *current rule*. L'exécution de la règle active peut modifier les variables globales et mettre à jour les ensembles *Current*, *Next*, *Completion*.

Voici un exemple de programme RUSSEL :

```
/*
 * show_bsm
 *
 * Prints the contents of an audit trail
 *
 * Aziz Mounji, Jan 1997
 *
*/

uses show_current;

global nb: integer;

rule show;
begin
     trigger off for_current show_current;
     trigger off for_next show;
     nb := nb + 1
end;

rule echo;
begin
     println( '-------------------------');
     println( 'total is: ', nb);
     println( '-------------------------')
end;

init_action;
begin
     nb := 0;
     trigger off for_next show;
     trigger off at_completion echo
end.
```

1.5. L'interface utilisateur d'ASAX

(Réf. 6)

Nous allons présenter l'interface en mode commande fourni par ASAX. En mode commande, pour analyser un programme en input, on doit spécifier :

1. le mode d'exécution de l'exécutable ASAX (mode standard, conversion ou mixte). Pour l'exécutable ASAX de type standard, le mode d'exécution est implicite

2. le data description file correspondant au fichier donné en input pour l'analyse

3. le module contenant le programme d'analyse

4. le fichier à analyser donné en input. Quand le mode d'analyse est standard, le fichier à analyser est un fichier NADF.

Pour générer un exécutable ASAX, on utilise la commande **masax**.

masax [-sc] [-lib Clib] [-fa io-obj] [asax-executable-file]

Deux composantes peuvent être modifiés dans ASAX : les routines des librairies C et les routines I/O virtuels pour le fichier native à analyser. Les routines des librairies C et les routines I/O du fichier à analyser sont liés au kernel ASAX pour générer un nouvel exécutable.

Les options :

-s la version ASAX générée par cette option suppose que le fichier à analyser est dans le format NADF. C'est l'option par défaut.
-c la version ASAX générée par cette option effectue l'analyse en mode conversion/evaluation. Dans ce mode, les enregistrements sont lus à partir du fichier à analyser et sont convertis au format NADF avant d'être soumis à l'évaluateur pour l'analyse. Avec cette option, l'utilisateur est supposé fournir trois routines virtuels au fichier native. Le module objet pour ces routines est spécifié en utilisant l'option **–fa**.
-sc la version ASAX fournie par cette option effectue l'analyse en mode mixte lequel supporte à la fois les modes standard et conversion/évaluation. Avec cette option, l'utilisateur est supposé fournir trois routines I/O au fichier native. Le module objet pour ces routines est spécifié en utilisant l'option **–fa**.
-lib Clib génère la version ASAX qui supporte les librairies C dont la représentation interne est identifée par le préfixe **C-lib**.
-fa io-obj génère une version ASAX où les routines virtuels I/O (vopen(), vread(), et vclose()) sont dans le fichier objet **io-obj.o**
Le fichier exécutable généré est **asax-executable-file** si un tel nom est donné en argument.

Les commandes ASAX

Syntaxe d'une commande :

asax-executable-file [-sclg] addf russel-program [audit-file]

Description :

asax-execuable-file est le fichier exécutable ASAX, **addf** est l'audit data description file, **russel-program** est le programme RUSSEL qui a **.asa** comme extension et **audit-file** est le fichier à analyser. **asax-executable-file** peut opérer en trois modes : mode standard (option **-s**), mode conversion/évaluation (option -c) et le mode mixte (option **–sc**) en fonction du format du fichier à analyser. Si **audit-file** est dans le format NADF, **asax-executable-file** est supposé être en mode standard ou en mode mixte. Mais si **audit-file** n'est pas dans le format NADF, **asax-executable-file** est supposé être en mode conversion/ealuation ou en mode mixte. Dans ce dernier cas, l'option **–c** doit être fournie.

Options :

-l seulement un check syntaxique du programme RUSSEL russel-program ; tous les modules utilisés par russel-program sont ignorés. Avec cette option le fichier audit en argument est ignoré.

-g effectue seulement un check syntaxique du programme RUSSEL russel-program ainsi que tous les autres modules utilisés par russel-program. Avec cette option le fichier audit donné en argument est ignoré.

-c asax-executable-file est supposé être en mode mixte. Le check syntaxique du programme RUSSEL est effectué et si aucune erreur n'est trouvée, le fichier audit est analysé en mode en mode conversion/évaluation.

-s c'est l'option par défaut. asax-executable-file est supposé être en mode standard ou mixte. Le check syntaxique du programme RUSSEL est effectué et si aucune erreur n'est trouvée, le fichier audit au format NADF est analysé en mode standard.

CHAPITRE 2 : LE LANGAGE TCL/TK

(Réf. 7)
Ce chapitre donne une synthèse des différentes notions sur le langage Tcl/Tk que nous avons utiliser dans la réalisation de ce mémoire. Il ne s'agit donc pas de présenter toutes les possibilités de programmation offertes par le langage. Plusieurs exemples sont fournis enfin d'illustrer les différentes notions que nous expliquons.

2.1. Historique

Tcl/Tk est né en 1988 et a été mis au point par John K.OUSTERHOUT de l'Université de Californie à Berkley. Développant un certain nombre d'outils interactifs, principalement pour la conception de circuits intégrés, il était fort ennuyé de devoir passer beaucoup de temps à développer des langages utilisés pour la commande de ces outils ; en plus chaque outil devait avoir son propre langage de commande. C'est pourquoi, il eut l'idée de concevoir un langage qui peut être utilisé dans différentes applications pour implémenter différentes choses. Un langage qui serait un package de librairies C pouvait convenir. Il devait être extensible pour que chaque outil puisse y ajouter des faits qui répondent à ses besoins spécifiques. Le résultat fût Tcl. Tk est né dans la perspective de permettre le développement des composants interactifs d'une application séparément. Ces composants pouvaient être rassemblés après en utilisant les scripts Tcl.
Le langage Tcl/Tk n'est pas seulement utilisé pour programmer des interfaces graphiques, il peut être aussi utilisé pour de nombreuses autres applications par exemple les bases de données, la communication réseau etc. Les applications Tcl/Tk peuvent être étendues aisément avec du code C.

2.2. Le langage Tcl

Pour introduire, voyons comment fonctionne Tcl. Le langage Tcl est défini par un interpréteur qui parse les commandes Tcl et par un ensemble de procédures qui exécutent les commandes. L'interpréteur et ses règles de substitution sont fixes mais de nouvelles commandes peuvent être crées de même que d'anciennes commandes peuvent être remplacées. Les structures de contrôle, les procédures, les expressions sont implémentés comme des commandes. Pour comprendre le fonctionnement de l'interpréteur Tcl, considérons la commande suivante qui est équivalent à la boucle while :

```
while {$nbr>5} {
   set result [expr $nbr*$nbr]
      }
```

Lorsque cette commande est exécutée, l'interpréteur Tcl sait uniquement qu'elle est composée de trois mots, le premier étant le nom de la commande. Une fois que la commande a été parsée, l'interpréteur passe les mots à la commande while qui va traiter son premier argument comme une expression et son deuxième argument comme un script. Si l'évaluation de l'expression donne un résultat non nul, alors while renvoie de nouveau le second argument à l'interpréteur Tcl pour l'évaluation. Cette fois-ci, l'interpréteur traite son argument comme un script, il effectue des substitution et appelle la commande set et expr.
Toute les commandes seront traitées de la même facon sauf qu'elles auront un nom différent. L'interpréteur appellera les procédure correspondantes pour exécuter ces commandes.
Le langage Tcl consiste donc en un certain nombre de règles simples pour parser les arguments et effectuer les substitutions. Pour apprendre Tcl, il faut d'abord connaître la syntaxe et les règles de substitution ; ensuite les commandes Tcl.

Syntaxe

Un **script Tcl** = une séquence de commandes.
Les commandes sont séparées par un passage à la ligne ou par un point-virgule.

Une **commande Tcl** = un ou plusieurs mots (*words* en anglais).

Le premier mot est le nom de la commande et le reste constitue son argument. Les mots sont séparés par des espaces. Chaque commande de l'exemple ci dessous est constituée de trois mots (La commande *set* est une affectation, elle sera expliquée ultérieurement).

Exemple

- *set a 22; set b 33*

ou

- *set a 22*
- *set b 33*

Evaluation d'une commande :

Division des responsabilités : Tcl évalue une commande en deux étapes: le **parsing** et l'**exécution**. Dans l'étape du parsing, l'interpréteur Tcl applique les règles pour trouver les commandes dans les mots et exécute les substitutions ; il n'interprète pas la valeur des mots. Pendant l'étape d'exécution, Tcl interprète la valeur des mots et produit un résultat de type string. **Tout est string en Tcl**. Le parseur n'assigne aucune signification à ses arguments.
Pour illustrer ceci comparons C et Tcl:

Exemple

C : *x = 4; y = x + 10*
y vaut 14

Tcl : *set x 4; set y x + 10*
Y vaut "x + 10"

Les commandes par contre assignent différentes significations à ses arguments :

Exemple

set a 122 La commande set considère son premier argument comme une variable et le second comme la valeur de cette variable. Il est aussi valable d'écrire set 122 a où 122 devient une variable et sa valeur est a.

Substitution

Tcl fournit trois sortes de substitutions : substitution de variables, de commandes et de backslash. Une substitution provoque le remplacement de certains mots par leurs valeurs.

- **Substitution de variables**

Syntaxe : $varName
Un nom de variable est constitué de lettres, digits ou d'underscores. La substitution peut se passer n'importe où dans un mot.

Exemple

set b 66 *66*

set a $b+$b *66+66*
set a $b4 *no such variable*

- **Substitution de commandes**

Syntaxe : [script]
Une substitution de commande cause le remplacement d'une partie ou de tout un mot par le résultat de la commande. La substitution peut se passer n'importe où dans un mot.

Exemple
set b 8 *8*
set a [expr $b+2] *10*
Les crochets et les caractères inclus sont remplacés par le résultat du script. Ainsi le résultat de expr $b+2 (10) devient le second argument de la commande set dans l'exemple ci-dessus.

- **Substitution de backslashes et contrôle de la structure des mots**

- La substitution de backslashes sert à insérer des caractères spéciaux dans un mot comme un passage à la ligne, [, $ sans qu'ils eussent une signification particulière pour le parseur.
- Les guillemets entourent une chaîne de caractères.
- Les accolades entourent une chaîne et empêche la substitution.
- Une ligne commençant par le signe # constitue un commentaire.

Exemple

set c 4 ; set b 11
set a world\ with\ $\ and\ space ⇒ *world with $ and space*
set a "c is $c ; b is $b" ⇒ *c is 4 ; b is 11*
set a { [expr $b*$c] } ⇒ *[expr $b*$c]*
 # Ceci est un commentaire.

2.2.1. Les structures de base

Les variables

Tcl supporte deux types de variables : les variables simples et les tableaux associatifs.
Les variables simples ont à la fois un nom et une valeur. En pratique, les noms commencent par une lettre et sont formés d'une combinaison de lettres, digits et d'underscores. Tcl distingue les majuscules et les minuscules :
filename n'a pas la même signification que *FileName*.
Les variables sont créées, lues et modifiées avec la commande *set*. Cette commande prend un ou deux arguments, le premier est le nom de la variable et le second si présent est la nouvelle valeur pour cette variable. La commande *unset* permet de supprimer une variable. Pour modifier les valeurs d'une variable, on utilise la commande *incr* et la commande *append*. *incr* prend deux arguments, le nom de la variable et un entier, il ajoute cet entier à la valeur de la variable. *append* prend une variable et un texte en argument et ajoute le texte à la variable.
En plus de simples variables, Tcl fournit les tableaux. Un tableau est une collection d'éléments, chaque élément est une variable et possède un nom ainsi qu'une valeur. Le nom d'un tableau est composé de deux parties : le nom du tableau et le nom d'un élément du tableau. Le nom du tableau et celui de l'élément peuvent être des strings. C'est pour cela qu'on parle de tableaux associatifs pour les distinguer des tableau dans d'autres langages où le nom du tableau doit être un entier.

Exemple

```
set prix(café) 33
set prix(limonade) 50
```

prix est le nom du tableau, café et limonade sont des élément du tableau.

Les expressions

Les expressions sont fort semblables à celle du C, avec un traitement avancé des strings. Les expressions combinent les valeurs et les opérateurs pour produire de nouvelles valeurs. La plupart des commandes Tcl attendent en argument une expression. La plus simple est la commande ***expr*** qui évalue ses argument comme des expressions et retourne un résultat de type string.
La substitution de commandes et celle de variables a lieu dans un expression. On peut utiliser d'autres commandes dans la commande *expr*.

Exemple

set b 5	*5*
*expr ($b*4) - 3*	*17*
expr $b * cos(2*$b)	*-4.19536*

Les opérandes d'une expression sont normalement des entiers ou des réels. Les entiers sont habituellement dans une notation décimale, mais si le premier caractère est 0, le nombre est lu en système octal et si les deux premiers caractères sont 0x, le nombre est lu en système hexadécimal.

Opérateurs arithmétiques

Les opérateurs arithmétiques sont : +, -, *, /, % (opérateur modulo). « - » peut être utilisé comme opérateur binaire pour la soustraction, ou comme opérateur unaire pour la négation.

Opérateurs relationnels

Les opérateurs relationnelles sont : <, <=, >=, >, ==, !=. Ils sont utilisés pour comparer deux valeurs et retournent une valeur booléenne (1 si la condition est vérifiée, 0 sinon).

Les opérateurs logiques

Les opérateurs logiques sont : && (AND), || (OR), ! (NOT). Ils sont utilisés pour combiner les résultats des opérateurs relationnels.

Les opérateurs bitwise

Ces opérateurs sont : &, |, ^, <<, >>, ~. Les opérandes pour ces opérateurs sont des entiers.

Les opérateurs de choix

L'opérateur ternaire ? : est utilisé pour sélectionner un résultat parmi deux :
expr {($a <$b) ? $a : $b} : Cette expression retourne la plus petite valeur entre $a et $b.
L'opérateur de choix examine si la valeur de son premier opérande est vrai ou faux. S'il est vrai, le résultat sera l'argument qui suit « ? » ; s'il est faux, c'est le troisième argument qui sera le résultat.

Les listes

Une liste est une collection ordonnée d'éléments où chaque élément peut avoir une valeur de type string, tel qu'un nombre, un nom d'une personne, un nom d'une fenêtre,.. Les listes sont des strings avec une structure particulière; on peut affecter une liste à une variable, passer une liste à une commande, l'inclure comme élément d'une autre liste.
Sous une forme simple, une liste est composée de zéro ou plusieurs éléments séparés par un espace.
Pour le regroupement, on utilise les accolades.

Exemple

rouge orange noire
a b {c d e} f

Les commandes pour les listes sont :

- Création des listes :

concat : cette commande prend n'importe quel nombre de listes, concatènent tous les éléments en une seule liste.
liste : cette commande combine ses arguments de telle façon que chaque argument devient un élément distinct de la liste résultante.
llength : cette commande renvoie le nombre d'éléments d'une liste.

- Modification des listes

linsert : cette commande ajoute un ou plusieurs éléments à une liste
lreplace : cette commande suppriment les éléments d'une liste et optionnellement ajoute de nouvelles éléments à leur place.
lrange : cette commande extraie les éléments d'un certain rang dans une liste.
lappend : cette commande permet d'ajouter de nouveaux éléments à une liste stockée dans une variable.

- Recherche dans une liste

lsearch : cette commande cherche un élément d'une valeur particulière dans une liste. La commande retourne l'indice de l'élément si celui-ci se trouve dans la liste, -1 sinon.

- Tri des listes

lsort : cette commande permet de trier les éléments d'une liste suivant l'ordre lexicographique.

2.2.2. Les structures de contrôle

Ces structures ressemblent à celles du C ; Ce sont tout simplement des commandes qui reçoivent des scripts Tcl en argument.

- **La commande IF**

La command *if* évalue une expression conditionnelle, teste son résultat et exécute un script en fonction du résultat.

Exemple

```
if {$x < 0} {
        set x 0
}
```

On peut aussi inclure dans la commande *if* des clauses *elseif* avec un *else* à la fin si tous les tests ont échoué.

Exemple

```
if {$x < 0} {
...
} elseif {$x == 0} {
...
} elseif {$x == 1} {
...
} else {
...
}
```

- **Les boucles WHILE, FOR et FOREACH.**

La commande *while* prend deux arguments : une expression et un script Tcl. Il évalue l'expression et si le résultat est différent de zéro, elle exécute le script. Ceci jusqu'à ce que le résultat de l'expression soit évalué à zéro et la boucle se termine et renvoie une chaîne vide.
La commande *for* exécute son premier argument comme un script Tcl, et évalue l'expression. Si l'expression retourne une valeur non nulle, *for* exécute le corps de la boucle suivi de la réinitialisation et de la réévaluation de l'expression. Cette séquence se poursuit jusqu'à ce que l'expression soit évaluée à zéro. Si l'expression retourne zéro au premier test, le corps de la boucle n'est pas exécuté. *for* renvoie une chaîne vide comme résultat.
La commande *foreach* prend trois arguments : le premier est le nom d'une variable, le deuxième est une liste, le troisième est un script Tcl qui constitue le corps de la boucle. La commande exécute le corps de la boucle pour chaque élément de la liste dans l'ordre. Avant d'exécuter le corps de la boucle, *foreach* affecte la variable à l'élément suivant dans la liste.
La commande *foreach* renvoie également une chaîne vide.

- **Les commandes BREAK et CONTINUE**

Ces commandes aident à sortir partiellement ou totalement de la boucle et ne prennent aucun argument.
La commande *break* entraîne l'arrêt immédiat de la boucle.
La commande *continue* empêche l'exécution de l'itération courante et la boucle se poursuit en exécutant l'itération suivante.

- **La commande SWITCH**

La commande *switch* test l'égalité d'une valeur à un certain nombre de patterns et exécute le script correspondant au pattern qui correspond à la valeur testée. Il est possible d'obtenir le même résultat avec la commande *if elseif* mais la commande *switch* permet d'écrire un code plus compact. La commande *switch* supporte trois types de patterns et on peut faire précéder la valeur à tester par la forme que l'on désire.
ces formes sont : *-exact* la comparaison exacte ; *-glob* qui s'exécute de la même façon que la commande *string match* (cette commande implémente le pattern matching est sera expliquée ultérieurement) ; *-regexp* qui fait la correspondance avec une expression régulière. Si le dernier pattern de la commande switch est *default*, il fait la correspondance avec n'importe quelle valeur. Le script sera donc exécuté même la correspondance n'est réalisée pour aucun autre pattern. Si dans une commande switch, le script est un tiret (-), switch exécute le script correspondant au pattern suivant.

- **La commande EVAL**

La commande *eval* accepte un nombre variable d'arguments, les concatène (avec l'espace comme séparateur) et exécute le résultat comme un script Tcl.

Exemple

```
set cmd "set a 0 "
...
eval $cmd
```

Ce script met a à 0 lorsque on invoque eval.

L'usage le plus important de la commande *eval* est de provoquer un autre niveau du parsing. Le parseur Tcl exécute un seul niveau de parsing et de substitution lorsqu'il parse une commande. Le résultat d'une substitution n'est pas reparsé pour les autres substitutions. Parfois, on a besoin d'un autre niveau de parsing et la commande *eval* permet d'y arriver.

Exemple

```
set vars {a b c d}
foreach i $vars {
        unset $i
}
```

Ce script est correct mais la commande *unset* prend un nombre quelconque d'arguments, on peut chercher à exécuter *unset* au moyen d'une seule commande. Le script suivant ne produit pas le résultat attendu :

```
set vars {a b c d}
        unset $vars
```

La solution est fournit par *eval*.

```
set vars {a b c d}
eval unset $vars
```

eval concatène ses arguments en une seule commande "*unset a b c d*", qui sera évaluée.

- **La commande SOURCE**

La commande *source* prend un seul argument qui spécifie un nom de fichier ; elle exécute le contenu du fichier comme un script Tcl.

2.2.3. Les procédures

La commande *proc* définit une procédure. Le premier argument de *proc* est le nom de la procédure à créer, le second argument est une liste d'arguments de la procédure et le troisième argument est un script Tcl qui constitue le corps de la procédure.

Exemple

```
proc plus {a b} {expr $a+$b}
```

Il faut remarquer que *proc* est une commande Tcl ordinaire. Les arguments de *proc* sont traités de la même manière que n'importe quelle autre commande Tcl.
La commande *return* provoque le retour de la procédure sans exécuter complètement le script entièrement et l'argument de *return* sera le résultat de la procédure.

Exemple

```
proc fact x {
        if {$x <= 1} {
                return 1
        }
        expr $x * [fact [expr $x-1]]
}
fact 4
→ 24
fact 0
→ 1
```

Les variables locales et globales

Les variables locales sont des variables qui sont accessibles uniquement à l'intérieur de la procédure tandis que les variables globales sont accessibles de l'extérieur de la procédure. Une procédure référence les variables globales par la commande *global*.

Les arguments :

On peut considérer trois cas spéciaux d'arguments :

- Si la procédure n'a pas d'arguments, la liste d'arguments est spécifiée comme une chaîne vide.
- Le cas où la liste d'arguments est une liste des listes avec chaque sous liste qui constitue un simple argument est traité comme suit : si la sous liste a un seul élément, cet élément est le nom de l'argument. Si la sous liste a deux éléments, le premier est le nom de l'argument et le second est sa valeur par défaut.

Exemple

```
proc inc {value {increment 1}} {
        expr $value + $increment
                }
```

Le premier élément value *dans la liste d'arguments spécifie le nom sans valeur par défaut ; le deuxième élément indique un argument de nom* increment *qui vaut 1 par défaut.* inc *peut alors être appelé avec un ou deux arguments.*

- Les procédures peuvent avoir un nombre variable d'arguments si le dernier élément de la liste d'arguments est le mot « *args* ». Les arguments qui précèdent *args* seront traités normalement puis les arguments additionnels seront affectés à *args*.

Call by reference : upvar

La commande *upvar* permet de faire un passage par référence. Par exemple dans le cas d'un tableau, *upvar* est utilisé pour accéder aux éléments du tableau. Le premier argument de *upvar* est le nom d'une variable accessible lors de l'appel de la procédure qui peut être locale ou globale ; le second argument est le nom d'une variable locale.

Exemple

```
proc tarif {prix} {
        upvar $prix a
        foreach el [lsort [array names a]] {
                puts " $el = $a($el) "
        }
}
set produit(pepsi) 40
set produit(coca) 40
set produit(biere) 30
tarif produit
→ biere = 30
  coca = 40
  pepsi = 40
```

La commande array names *permet d'accéder à la liste des éléments du tableau.*

2.2.4. La gestion des chaînes de caractères

Tout est string en Tcl. Il n'est alors pas surprenant qu'il y ait plusieurs commandes pour manipuler les chaînes de caractères.
Les expressions régulières proviennent de la théorie des langages formels. Dans cette théorie, un langage est un ensemble de chaînes de caractères ; d'une part, on définit le langage, d'autre part, on développe des algorithmes pour reconnaître les chaînes de caractères qui font partie du langage défini. En pratique, on ne cherche pas à déterminer si une chaîne de caractères appartient au langage mais on cherche à déterminer les sous-chaînes qui correspondent à des expressions régulières (c'est ce qu'on appelle *match* en anglais). L'expression régulière auquel correspond la sous-chaîne est appelée *pattern*.

Glob-style pattern matching

La plus simple forme de pattern matching s'appelle *glob-style*. Cette forme est implémentée par la commande *string match*.

Exemple

```
set new { }
foreach el $list {
        if [string match Tcl* $el] {
                lappend new $el
        }
}
```

Cette procédure extraie tous les mots commençant par « Tcl ».

Le pattern matching avec les expressions régulières

Les expressions régulières sont construites à partir des atomes. Une expression régulière est constituée par un ou plusieurs atomes. Dans la plupart des cas, les atomes sont des simples caractères.
Un certain nombre de caractères ont une signification spéciale parmi les expressions régulières.

- « ^ » et « $ » sont des atomes qui correspondent respectivement au début et à la fin de la chaîne d'entrée : ^abc correspond à n'importe quelle chaîne de caractères commençant par abc ; abc$ correspond à n'importe quelle chaîne de caractères se terminant par abc.
- « . » code pour un caractère simple.
- « \x » code pour le caractère x, où x est un simple caractère.
- « *[chars]* » code pour n'importe quel simple caractère inclus dans *chars*. Si le premier caractère de *chars* est « ^ », alors l'expression code pour n'importe quel simple caractère qui ne se trouve pas dans *chars*. Une séquence de la forme a-b dans *chars* code pour tous les caractères ASCII compris entre a et b.
- « * » code pour une séquence de 0 ou plusieurs atomes
- « + » code pour une séquence 1 ou plusieurs atomes
- « ? » code pour une chaîne vide ou pour un atome
- « | » est utiliser pour l'union de deux expressions régulières. L'expression résultante code pour l'une ou l'autre des deux expressions entourant le « | ».

C'est la commande *regexp* qui invoque la correspondance avec une expression régulière. Sous sa forme la plus simple, il prend deux arguments : un pattern et une chaîne d'entrée et renvoie 1 si la chaîne d'entrée correspond à l'expression régulière, 0 dans l'autre cas.

NB : Le pattern doit être inclus entre les accolades. Ceci pour que les caractères tels que « $ », « [», «] » soient passés à la commande ***regexp***.
Il existe deux options pour la commande *regexp* : *-nocase* : qui spécifie que tous les caractères de la chaîne seront convertis en minuscules avant de faire la comparaison. *–indices* : cause que toute variable additionnelle contient une liste de deux nombres, l'un donne le premier indice dans le pattern de la sous-chaîne qui y correspond et l'autre donne le dernier indice de la sous-chaîne.

Exemple

regexp –indices {[a-z]ab} « abab » Match
Après cette commande, la variable Match aura les valeurs 1 et 3.

La commande *regsub* est utilisée pour faire les substitutions avec des expressions régulières. Le premier argument est une expression régulière, le deuxième argument est une chaîne de caractère. *regsub* ne vérifie pas seulement la correspondance mais il crée un nouveau string en effectuant une substitution. Le pattern de substitution est le troisième argument de la commande ; le nouveau string est affecté à une variable qui est le dernier argument de *regsub*.

Exemple

regsub a ababa zz x
Après cette commande x aura comme valeur zzbaba.

La commande *regsub* possède deux options : -nocase qui provoque la conversion de tous les caractères de la chaîne en entrée en minuscule ; -all qui provoque la substitution de toutes les correspondances trouvées.
La commande *format* permet d'effectuer la conversion de format dans les strings. L'argument en entrée est un string dont le format de conversion doit être spécifié. Pour chaque conversion, la commande format effectue un remplacement suivant le format souhaité.
Les formats supportés par la commande format sont les même que ceux définis pour ANSI C *sprintf*, comme par exemple *%f* pour les nombre réels, *%d* pour les entiers décimaux, *%x* pour les entiers hexadécimaux, *%e* pour les nombres réels avec mantisse exponentielle.

Exemple

format « La racine carrée de 10 est %.3f » [expr sqrt (10)]
Le résultat sera La racine carrée de 10 est 3.162.

La commande *scan* est l'inverse de la commande *format*. Cette commande reçoit un string formaté, le parse sous le contrôle d'un string de format, extraie les champs correspondant à la conversion spécifiée dans le string de format et place les valeurs extraites dans les variables Tcl. Le premier argument de la commande *scan* est le string à parser, le second argument est le string de format qui contrôle le parsing, les autres arguments additionnels sont des noms des variables qui contiendront les valeurs converties.
Cette commande est utilisée pour parser les chaînes de caractères simples. Une autre utilisation de cette commande est la conversion des caractères ASCII à leurs valeurs entières, dans ce cas le format est spécifié avec *%c*.

Exemple

scan « 1lire, 40.25 Euro » « %d livre, %f » a b
après l 'exécution de cette commande la variable a aura la valeur 1,
b aura la valeur 40,25.

2.2.5. La gestion des fichiers

Les commandes de gestion de fichiers permettent la lecture et l'écriture des fichiers, séquentiellement ou d'une façon aléatoire. On peut extraire des information du système sur les fichiers, par exemple, le temps du dernier accès au fichiers, ...
Les noms de fichiers sont spécifiés en utilisant la syntaxe UNIX. Un fichier nommé x/y/z est un fichier qui se trouve dans le répertoire y et qui à son tour se trouve dans le répertoire x. Un fichier nommé ~lgumiro/mbox réfère à un fichier nommé mbox se trouvant dans le répertoire home de l'utilisateur lgumiro.

Les commandes I/O de base pour les fichiers

La commande *open* permet l'ouverture d'un fichier. Elle prend deux arguments : le nom du fichier à ouvrir et le mode d'accès. Les différents modes d'accès peuvent avoir les valeurs suivantes :

- *r* ouvre un fichier en lecture seule ; le fichier doit exister.
- *r+* ouvre un fichier en lecture et en écriture ; le fichier doit exister.
- *w* ouvre un fichier en écriture seule ; si le fichier existe, il est écrasé, sinon la commande crée un nouveau fichier.
- *w+* ouvre un fichier en lecture et en écriture ; si le fichier existe, il est écrasé, sinon un nouveau fichier est créé.
- *a* ouvre un fichier en écriture seule et se positionne initialement en fin de fichier. Si le fichier n'existe pas, un nouveau fichier est créé.
- *a+* ouvre un fichier en lecture et en écriture et se positionne initialement en fin de fichier. Si le fichier n'existe pas un nouveau fichier est créé.

La commande *open* renvoie l'identificateur du fichier ouvert. Trois identificateurs de fichier sont prédéfinis et sont toujours disponibles ; il s'agit de *stdin* qui identifie le fichier d'entrée standard, *stdout* qui identifie le fichier de sortie standard et *stderr* qui identifie le fichier erreur standard.
La commande *gets* lit une ligne du fichier ouvert en ignorant le caractère de fin de ligne, place le contenu de la ligne dans une variable et renvoie la somme du nombre de caractères stockés dans cette variable. Cette commande prend donc deux arguments : le nom du fichier et le nom d'une variable. Si la commande atteint la fin du fichier sans avoir lue aucun caractère, elle place une chaîne vide dans la variable et renvoie la valeur –1. Si le nom d'une variable n'est pas fourni en argument, la commande renvoie le contenu de la ligne comme résultat.
La commande *read* lit tout le contenu du fichier et renvoie le nombre de bytes lus.
La commande *puts* permet d'écrire les données dans un fichier. Elle prend deux arguments, le nom du fichier et la chaîne de caractère à écrire. *puts* place un caractère de fin de ligne à la fin de la chaîne de caractère. Dans la plupart des cas *puts* place les valeurs lues en mémoire ; ce n'est que lorsqu'une quantité suffisante de données est stockée que l'écriture a lieu en une seule opération.
La commande close permet de fermer le fichier ouvert et de libérer ainsi les ressources qui lui ont été alloués.
La commande *flush* permet d'écrire les données dans un fichier. Cette commande n'utilise pas de mémoire tampon, elle écrit immédiatement dans le fichier.
Toutes ces commandes que nous venons de voir permettent un accès séquentiel aux fichiers.

L'accès aléatoire aux fichiers

Pour un accès aléatoire, on spécifie pour chaque fichier ouvert, la position d'accès qui détermine l'endroit où la prochaine lecture ou écriture aura lieu dans le fichier. Lorsqu'un fichier est ouvert, la position d'accès est au début ou à la fin du fichier selon le mode spécifié

avec la commande *open*. Après chaque opération de lecture ou d'écriture, la position d'accès est augmentée du nombre de bytes transférés.
La commande *seek* peut être utilisée pour modifier la position courante d'accès. Cette commande prend deux arguments, le nom du fichier et un entier qui mesure le déplacement dans le fichier. Elle peut aussi avoir un troisième argument qui spécifie l'origine du déplacement. Ce troisième argument peut être :

start qui a le même effet que comme si l'argument était omis
current qui mesure déplacement par rapport à la position courante d'accès
end qui mesure le déplacement à partir de la fin du fichier.

La commande *tell* donne la position d'accès courante pour un nom de fichier particulier.
La commande *eof* permet d'indiquer si la fin du fichier est atteinte. Elle renvoie 0 ou 1 selon le cas. Elle prend en argument le nom du fichier concerné.

La gestion des noms de fichiers

Il existe deux commandes pour manipuler les noms de fichiers.
La commande *glob* prend en argument un ou plusieurs patterns et renvoie la liste des noms de fichiers qui correspondent aux patterns. Cette commande utilise les règles de correspondance de la commande *string match* que nous avons présentée précédemment. Si l'argument de *glob* se termine par un slash alors la correspondance est effectué uniquement avec les noms de répertoire. L'option *-nocomplain* évite de renvoyer une erreur si le nom du fichier renvoyé par *glob* est une chaîne vide.

Exemple

*glob *.c *.h*
Cette commande renvoie tous les fichier qui ont l'extension .c ou .h dans le répertoire de travail.

La deuxième commande est la commande *file* qui possède un grand nombre d'options pour manipuler les noms de fichiers et extraire les informations sur ces fichiers.

Exemple

- *La commande* ***file extension*** *renvoie l'extension du nom du fichier ;*
  file extension src/main.asa renvoie .asa
- *La commande* ***file rootname*** *enlève l'extension du nom de fichier*
  file rootname main.asa renvoie main
- *cd dir*
  *foreach file [lsort [glob -nocomplain *]] {*
  if { [file isdirectory $file] } {
  $op.frame2.list insert end "$file/"
  ...

Cette commande trie par ordre alphabétique tous les noms dans le répertoire dir *et vérifie si ce sont des noms de répertoires.*

Pour gérer le répertoire courant, Tcl fournit deux commandes :

- La commande *pwd* permet d'afficher le répertoire courant. Elle n'a pas d'argument.
- La commande *cd* permet de changer le répertoire de travail. Elle prend un simple argument et change le répertoire courant à la valeur de cette argument. Sans argument, cette commande change le répertoire courant au répertoire home de l'utilisateur.

2.2.6. La gestion des processus

Tcl fournit un certain nombre de commandes pour gérer les processus.

Création des processus

La commande *exec* permet de créer un nouveau processus. Le premier argument de la commande *exec* est le nom du programme à exécuter, chaque autre argument supplémentaire est l'argument du processus créé. La commande *exec* collecte toutes les informations écrites à la sortie standard par le processus et renvoie ces informations comme résultat.

Exemple

exec wc /usr/include/stdio.h
Cette commande renvoie par exemple 71 230 1731 /usr/include/stdio.h

La commande *exec* supporte les redirections de la même manière que dans la programmation shell UNIX.

Exemple

exec cat << « test data » > toto
Cette commande passe à cat le string test data en entrée qui le redirige dans le fichier toto.

La commande *exec* permet également de créer un *pipe* entre processus en utilisant le caractère « | ».
Si le dernier argument de *exec* est le caractère « & », le processus est exécuté en mode *bacground.*
exec se comporte de la même façon que dans la programmation shell UNIX avec cette différence importante : *exec* ne s'applique pas sur les extensions des noms de fichiers.

Exemple

*exec rm *.o*
*Cette commande renvoie un message d'erreur. Pour réussir cette commande, on pourrait combiner les commandes eval et glob comme suit : eval exec rm [glob *.o].*

La commande *open* permet également de créer des processus. Si le premier caractère du nom de fichier passé à la commande *open* est le caractère « | », l'argument spécifie la création d'un *pipe*. Le reste de l'argument après le symbole « | » a exactement la même signification que l'argument de la commande *exec*.

Exemple

set f [open |prog r+]
Cette commande crée un pipe en lecture et en écriture pour prog. Après cette commande, on peut utiliser puts pour écrire dans le pipe et gets pour y lire les données.

Identification des processus

Il existe trois façon pour accéder à l'identification des processus en Tcl.

- Lorsque on crée un pipe en mode background en utilisant la commande *exec*, celle-ci renvoie une liste comprenant tous les processus dans le pipe.
- On peut également exécuter la commande *pid* sans argument qui renvoie l'identifiant du processus courant.
- Enfin, on peut exécuter la commande *pid* avec un nom de fichier en argument. S'il y a un pipe pour ce fichier ouvert, la commande *pid* renvoie les identifiant des processus dans le pipe.

Exemple

set file_rst [open "asaxn addf.bsm $SCE $DDF" r]
pid $file_rst
→ 830

La variable *env* contient toutes les variables d'environnement. Toute modification de *env* se répercute sur les variables d'environnement de toutes les processus et les nouvelles valeurs obtenues seront passées aux processus fils créés avec la commande *exec* ou *open*.

Exemple

open_dir "$env(HOME)"
"$env(HOME)" est le répertoire home de l'utilisateur.

La commande *exit* permet de terminer un processus. Cette commande peut avoir en argument une valeur entière optionnelle. La valeur 0 signifie une terminaison normal du processus et une valeur différente de 0 signifie une terminaison anormale.

2.2.7. Résumé de la syntaxe Tcl

- **Script** = commandes séparés par un passage à la ligne ou par un point-virgule.
- **Commande** = mots séparés par un espace.
- $ cause la substitution de variables.
- [] cause la substitution de commandes.
- « » entoure l'espace blanc.
- { } entoure tous les caractères spéciaux.
- \ précède le caractère suivant.
- # est utilisé pour les commentaires au début d'une commande.

2.3. Création d'un interface graphique avec Tk

2.3.1. Introduction

Tk est une extension du Tcl qui fournit un ensemble de commandes pour programmer des interfaces graphiques. Il utilise une application nommée **wish** (**wi**ndowing **sh**ell) qui comprend toutes les commandes définies pour Tk. L'élément de base utilisé par Tk pour construire un interface graphique est l'*objet interactif concret* (Concrete Interactor Object, physical interactor, Control ou widget) que nous noterons dans la suite OIC.
Les OICs sont divisés en classes telles que buttons, menus, scrollbars. Tk fournit une seule commande pour chaque classe d'OICs qu'on appelle *class command.*

2.3.2. Structure d'une application Tk

Les OICs sont arrangés suivant une structure hiérarchique. Un OIC peut avoir un nombre quelconque de descendants dans l'arbre, et l'arbre peut avoir une profondeur quelconque. Les OICs qui ont une signification spéciale pour un utilisateur (par exemple button), constituent les feuilles de l'arbre des OICs. Les OICs qui ne sont pas des feuilles sont en général utilisés comme des conteneurs pour l'organisation et l'arrangement des OICs feuilles.

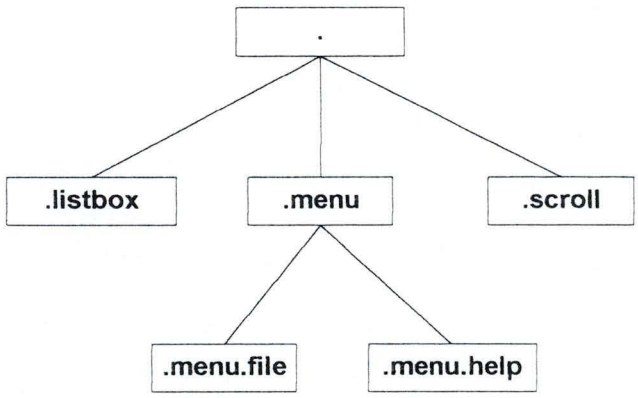


Fig. 2.1. Hiérarchie des OICs.

Les noms des OICs ont une hiérarchie similaire à celle des commandes UNIX pour les noms de fichiers sauf que le séparateur utilisé est le « . » au lieu du « / ».
« . » se rapporte à l'OICs du plus haut niveau dans la hiérarchie, c'est l'OIC principal.

Exemple

> *Le nom .a.b.c désigne l'OIC .c qui est le fils de l'OIC .a.b qui est à son tour le fils de l'OIC .a et ce dernier est le fils de l'OIC principal.*

Le terme application en Tk se rapporte à un arbre hiérarchique des OICs, un seul interpréteur Tcl associé à cet arbre et toutes les commandes fournies par cette interpréteur. Il existe un seul processus associé à chaque application.
L'OIC principal de chaque application occupe le plus haut niveau dans X window et est donc géré par le gestionnaire des fenêtres. La plupart des autre OICs d'une application utilisent des fenêtres internes ; Ces OICs sont appelés des OICs internes. Quelques classes d'OICs utilisent des fenêtres de plus haut niveau qui peuvent être contrôlées directement par le gestionnaire des fenêtres. Ce sont des OICs de plus haut niveau (top-level widgets). Leurs fenêtres sont des descendants de la fenêtre racine alors que les fenêtres des OICs internes sont des descendants des fenêtres des OICs parents dans l'arbre hiérarchique.

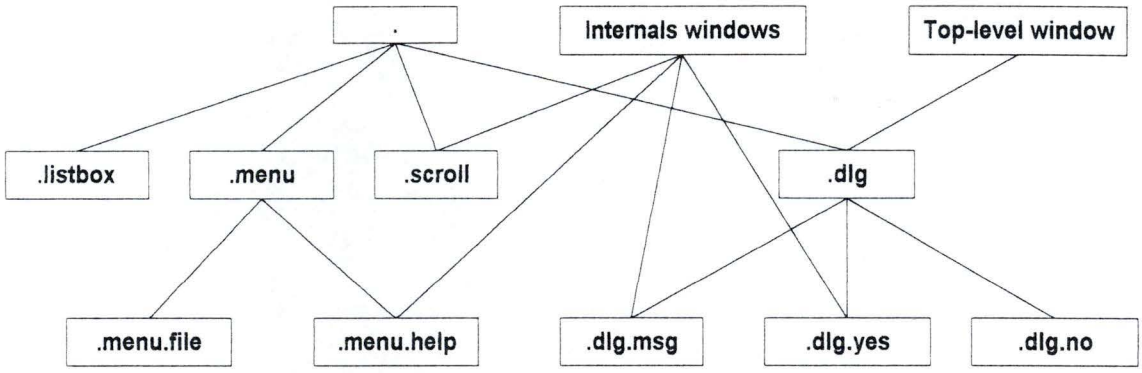


Fig. 2.2. Types de fenêtres dans l'arbre hiérarchique des OICs.

2.3.3. Les classes d'OICs fournies par Tk

Nous allons passer en revue les différentes classes d'OICs fournies par Tk. Chaque classe est définie par ses options de configuration, les commandes d'OICs et son comportement.
Les options de configurations représentent l'état de l'OIC, par exemple la couleur. Chaque OIC de l'application possède une commande qui est utilisée pour déclencher les actions dans l'OIC. L'action déclenchée par les commandes d'OIC varie d'une classe à l'autre. Le comportement des OICs spécifie l'exécution des scripts Tcl en réponse aux actions de l'utilisateur.
On présentera seulement un seul OIC (button) en détail, les autres on ne fera que les définir.

- Le button

Les options standards :

- activebackground : Command-Line Name
  aciveBackground : Database Name
  Foreground : Data Base Class

Spécifie la couleur de font à utilise lorsqu'un dessine l'élément actif. Un élément est actif lorsque la souri est positionnée au dessus de l'élément et la pression du bouton de la souri déclenchera un certain nombre d'actions.

- activeforeground : Command-Line Name
  activeForeground : Database Name
  Background : Data Base Class

Spécifie la couleur de premier plan à utiliser lorsqu'on dessine un élément actif.

- anchor : Command-Line Name
  anchor : Database Name
  Anchor : Data Base Class

Spécifie comment l'information sera affichée dans l'OIC. Les valeurs peuvent être **n**, **ne**, **e**, **se**, **s**, **sw**, **w**, **nw**, ou **center.**

- background ou bg : Command-Line Name
  background : Database Name
  Background :

Spécifie la couleur normal de font à utiliser lorsqu'on affiche l'élément.

- bitmap : Command-Line Name
  bitmap : Database Name
  Bitmap : Data Base Class

Spécifie le bitmap à afficher dans l'OIC. La façon dont le bitmap est affichée doit aussi être spécifiée par d'autres options telles que *anchor* ou *justify*.

- borderwidth ou bd : Command-Line Name
  borderWidth : Database Name
  BorderWidth : Data Base Class

Spécifie la valeur non-négative qui indique la largeur de la bordure externe en 3-D à dessiner autour de l'OIC. Cette valeur doit également être utilisée lorsqu'on dessine les effets 3-D à l'intérieur de l'OIC.

- cursor : Command-Line Name
  cursor : Database Name
  Cursor : Data Base Class

Spécifie le pointeur de la souri à utiliser pour l'OIC.

- disabledforeground : Command-Line Name
  disabledForeground : Database Name
  DisabledForeground : Data Base Class

Spécifie la couleur de premier plan à utiliser lorsqu'on dessine un OIC désactivé.

- font : Command-Line Name
  font : Database Name
  Font : Data Base Class

Spécifie le font à utiliser lorsqu'on dessine du texte à l'intérieur de l'élément.

- foreground : Command-Line Name
  foreground : Database Name
  Foreground : Data Base Class

Spécifie la couleur de premier plan à utiliser lorsqu'on affiche l'OIC.

- highightbackground : Command-Line Name
  highlightBackground : Database Name
  HighlightBackground : Data Base Class

Spécifie la couleur à utiliser dans la région transversale lorsque l'OIC n'a pas le focus d'entrée.

- highlightcolor : Command-Line Name
  highlightColor : Database Name
  HighlightColor : Data Base Class

Spécifie la couleur à utiliser dans le rectangle transversal qui est dessiné autour de l'OIC lorsqu'il a le focus d'entrée.

- highlightthickness : Command-Line Name
  highlightThickness : Database Name
  HighlightThickness : Data Base Class

Spécifie une valeur non-négative qui indique la largeur du rectangle lorsque l'OIC a le focus d'entrée.

- image : Command-Line Name
  image : Database Name
  Image : Data Base Class

Spécifie l'image à afficher dans l'OIC. Cette image doit être créée par la commande **image create**.

- justify : Command-Line Name
  justify : Database Name
  Justify : Data Base Class

Spécifie comment les lignes d'un texte sont disposées dans l'OIC. Les valeur peuvent **left**, **center** ou **right**.

- padx : Command-Line Name

  padX : Database Name

  Pad : Data Base Class

Spécifie une valeur non-négative qui indique l'espace à réserver autour de l'OIC dans la direction X.

- pady : Command-Line Name
  padY : Database Name
  Pad : Data Base Class

Spécifie une valeur non-négative qui indique l'espace à réserver autour de l'OIC dans la direction Y.

- relief : Command-Line Name
  relief : Database Name
  Relief : Data Base Class

Spécifie les effets 3-D possible pour l'OIC. Les valeurs acceptables sont : **raised**, **sunken**, **flat**, **ridge**, **solid**, et **groove.**

- takefocus : Command-Line Name
  takeFocus : Database Name
  TakeFocus : Data Base Class

Détermine si la fenêtre accepte la sélection à l'aide des touche du clavier.

- text : Command-Line Name
  text : Database Name
  Text : Data Base Class

Spécifie le texte à afficher à l'intérieur de l'OIC.

- textariable : Command-Line Name
  textariable : Database Name
  Variable :

Spécifie une variable. La valeur de la variable est le texte à afficher dans l'OIC. Si la valeur de la variable change, l'OIC se mettre à jour automatiquement et affichera la nouvelle valeur.

- underline : Command-Line Name
  underline : Database Name
  Underline : Data Base Class

Spécifie la valeur entière de l'indice du caractère à souligner dans l'OIC. La valeur 0 correspond au premier caractère.

- wraplength : Command-Line Name
  wrapLength : Database Name
  WrapLength : Data Base Class

Spécifie la longueur maximum de la ligne dans l'OIC. Les ligne qui dépassent cette longueur sont coupées et sont continuées à la ligne suivante.

Les options spécifiques à l'OIC :

- command : Command-Line Name
  command : Database Name
  Command : Data Base Class

Spécifie une commande Tcl associée au bouton. Cette commande est invoquée lorsque le bouton de la souri est lâché au dessus de l'OIC.

- default : Command-Line Name
  default : Database Name
  Default : Data Base Class

Spécifie un des trois états de l'anneau par défaut : **normal**, **active** ou **desabled**.

- height : Command-Line Name

height : Database Name
Height : Data Base Class
Spécifie la hauteur du bouton.

- state : Command-Line Name
  state : Database Name
  State : Data Base Class

Spécifie un des trois états du bouton : **normal**, **active**, **desabled**.

- width : Command-Line Name
  width : Database Name
  Width : Data Base Class

Spécifie la largeur désirée d'un bouton.

Les commandes de l'OIC :

- *pathName* **cget** *option* : cette commande renvoie la valeur courante de l'option de configuration donnée par *option*.
- *pathName* **configure** *?option ? ?value option value ...?* : cette commande modifie les options de configuration de l'OIC.
- *pathName* **flash** : cette commande fait clignoter le bouton. Cela se fait en affichant le bouton plusieurs fois alternativement entre la couleur normale et la couleur active.
- *pathName* **invoque :** cette commande appelle la commande Tcl associée au bouton.

Le comportement par défaut de l'OIC :

[1]

Un bouton s'active lorsque la souri passe au dessus de lui et se désactive lorsque la souri le quitte.

[2]

Le relief du bouton 1 passe à l'état creux (sunken) lorsque le bouton de la souri est pressé ; le relief retourne à l'état initial lorsque la souri est relâchée.

[3]

Si le bouton 1 est pressé et relâché après au dessus du bouton, celui-ci est invoqué. Si la souri est relâché lorsqu'elle n'est plus au dessus de la souris, aucune invocation n'a lieu.

[4]

Si le bouton a le focus d'entrée, l'espace clé provoque l'invocation du bouton.

- Le frame

Les frames sont typiquement utilisés comme des conteneurs pour regrouper les autres OICs. Elles apparaissent sous forme de région rectangulaire colorée. Les frames n'ont pas de comportement par défaut, ils ne doivent pas normalement interagir avec la souri ou le clavier.

- Le label

Les labels sont utilisés pour afficher du texte ou le bitmap.

- Le checkbutton

Les checkbuttons sont utilisés pour effectuer des choix binaires par exemple pour activer ou désactiver.

- Le radiobutton

Les radiobuttons sont utilisés pour sélectionner un parmi plusieurs choix mutuellement exclusifs.

- Le menu

Cet OIC permet de réaliser des menus.

- Le menubutton

Les menubuttons sont associés aux menus.

- Le message

Les OICs messages permettent d'afficher un texte de plusieurs lignes.

- L'entry

Ces OICs permettent saisir ou d'afficher une ligne de texte.

- Le text

Les texts permettent d'afficher ou de saisir un ou plusieurs lignes de texte.

- Le canvas

Ces OICs créent un surface de dessin ou l'on peut dessiner plusieurs types d'objets (rectangle, cercle, ...).

- Le scrollbar

Les scrollbars permettent de contrôler l'affichage dans d'autres OICs, par exemple les texts.

- Le scale

C'est un OIC qui permet d'afficher une valeur entière et permet d'éditer une valeur en déplaçant un bouton.

- Le listbox

C'est un OIC qui permet d'afficher une liste de strings et permet d'en sélectionner un ou plusieurs.

- Le toplevel

Les toplevels sont comme des frames sauf qu'ils occupent les fenêtres de plus haut niveau, alors que les frames occupent les fenêtres internes.

2.3.4. Le placement des OICs

Les gestionnaires de géométrie (geometry menager) déterminent la taille et la position des OICs. Ceux-ci ne déterminent pas leur propre géométrie. Les OICs ne peuvent apparaître à l'écran que s'il sont gérés par le gestionnaire de géométrie.

Le gestionnaire de géométrie arrange les OICs esclaves relativement à l'OIC maître. Le maître est en général le parent de la fenêtre esclave.

Le gestionnaire de géométrie reçoit trois sortes d'information qu'il utilise pour calculer la géométrie. La première information est que chaque OIC esclave demande une hauteur et une largeur particulière. C'est la dimension minimale nécessaire à l'OIC pour afficher ses informations. La deuxième information provient du concepteur de l'application et est utilisée pour contrôler l'algorithme qui détermine la géométrie. La nature de cette information varie d'un gestionnaire à l'autre. Par exemple, on peut demander que les trois OICs soient placés sur une ligne de gauche à droite dans l'OIC maître ; le gestionnaire de placement cherche la place nécessaire pour les trois OICs et les place sur une ligne de telle sorte que chaque OIC dispose de l'espace dont il a besoin.

La troisième information est la géométrie de l'OIC maître. On peut demander que l'OIC esclave soit placé dans le coin supérieur gauche par exemple.

Une fois que le gestionnaire de géométrie à reçu ces informations, il exécute un algorithme de réarrangement pour déterminer la largeur, la hauteur et la position de chaque OIC esclave. Certains gestionnaires de géométrie déterminent la taille nécessaire à l'OIC maître. Par exemple le packer calcule l'espace nécessaire de l'OIC maître en fonction des OICs esclaves comme spécifié par le concepteur de l'application. Le maître aura ces dimensions ce qui supprime les demandes qui avaient été faites par les OICs maîtres eux-memes.

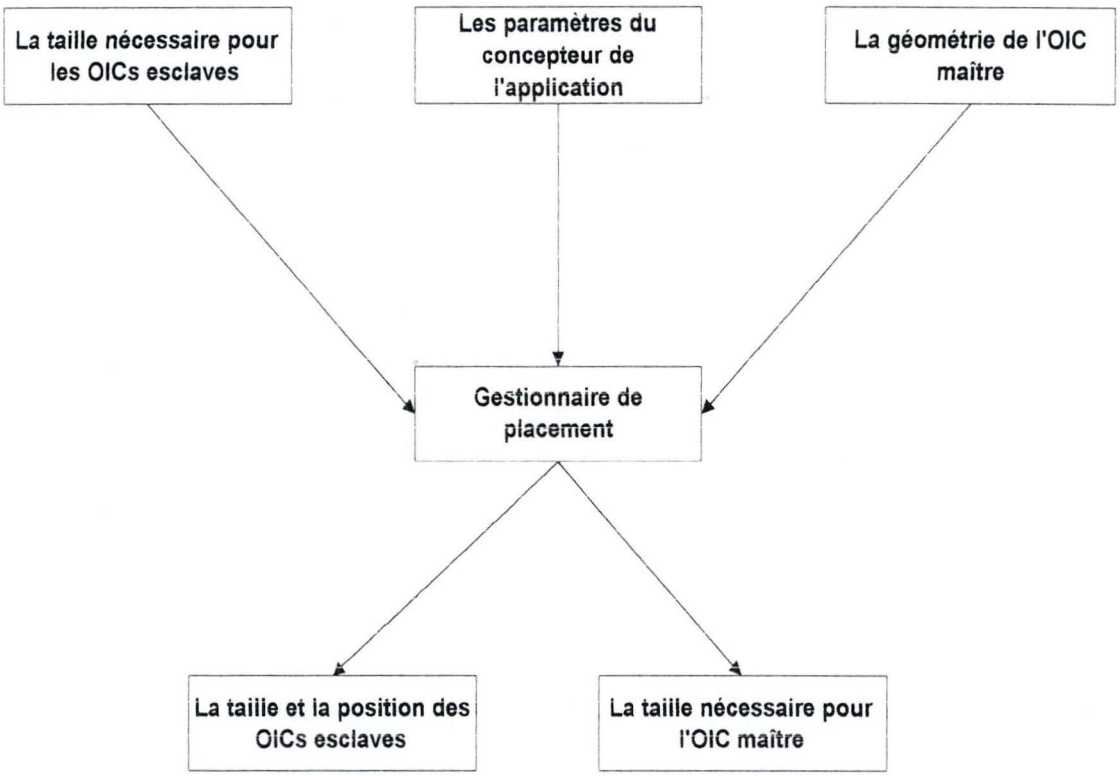


Fig. 2.3. Principes de fonctionnement du gestionnaire de géométrie.

Tk possède plusieurs gestionnaires de géométrie : le packer, le placer, le grid, le canvas.

Le packer

C'est le gestionnaire de géométrie le plus utilisé en Tk. Il arrange les OICs esclaves dans la fenêtre maître en plaçant un OIC à un moment donné dans la fenêtre maître.
Le packer maintient une liste d'OICs esclaves pour un OIC maître donné, c'est ce qu'on appelle *packing list*. Le packer arrange les OICs esclaves en suivant l'ordre dans la liste et en plaçant un OIC à chaque étape.
Les options du packer :

- Le padding :

Le padding permet de réserver de l'espace plus grand à l'OIC esclave. Il y a le padding externe et le padding interne. Le padding externe est spécifié avec les options *-padx* et *-pady*. La padding externe est spécifié avec les options *-ipadx* et *-ipady*.

- Le filling :

Cette option permet à l'OIC esclave de remplir toute sa parcelle si elle est plus grande que celle qui lui était nécessaire. Le filling est spécifié avec l'option *-fill {x,y,both}*.

- L'expanding :

Si l'OIC maître possède un espace plus large que celle nécessaire à ses OICs esclaves, on peut utiliser l'option *-expand* pour demander à un OICs particulier de remplir tout espace libre à l'intérieur de l'OIC maître.

- L'anchoring :

Si la parcelle est plus grande que celle nécessaire à l'OIC esclave, celui-ci sera placé au centre de la parcelle par le packer. L'option *-anchor* permet de placer l'OIC esclave dans une autre position.

- Sides :

L'option *-side* permet de spécifier la position où sera allouée la parcelle de l'OIC esclave dans l'espace disponible. Les valeurs possibles sont : *left, right, top, bottom.*

Le placer

Le placer un gestionnaire de géométrie qui permet de placé un OIC esclave dans une position absolue ou relative dans l'OIC maître. Une position absolue peut être par exemple, 1 cm de la limite coté haut de l'OIC maître, et 2 cm de la limite coté gauche.
Le placer considère les OICs esclave indépendamment et il est plus difficile de les arranger en lignes ou en colonnes.

Le grid

Le grid permet un arrangement en ligne et en colonnes. Il utilise l'option *-sticky* pour positionner les OICs dans la grille et peut utiliser les options *-rowspan* (pour fusionner les lignes) et *-colomnspan* (pour fusionner les colonnes).

Le canvas

Il est associé aux OICs canvas et permet de combiner les OICs avec les éléments graphiques tels que les lignes et du texte.

2.3.5. Bindings

C'est un mécanisme fournit par Tk qui permet de créer un handler pour un événement quelconque dans un OIC. Le binding associe un script Tcl à un événement X.
Par exemple, un OIC entry a un handler toujours prêt à prendre en charge un caractère lorsqu'un événement approprié du clavier se produit.
Un événement est généré par X pour signaler à une application une occurrence intéressante. Chaque événement à un type qui indique quelle sorte d'occurrence s'est produite. Citons quelques types d'événements qui sont communément utilisé dans le binding :

Key ou KeyPress	une touche du clavier est enfoncée
KeyRelease	une touche du clavier est lâchée
Button ou ButtonPress	un bouton de la souri est enfoncé
ButtonRelease	un bouton de la souri est lâché
Enter	le pointeur de la souri est dans l'OIC
Leave	le pointeur de la souri quitte l'OIC
Motion	le pointeur de la souri se déplace d'un endroit à l'autre dans un même OIC.

C'est la commande *bind* qui permet de créer, modifier et d'enlever le binding.
Lorsqu'une application est initialisée, il entre dans une *boucle d'événement* pou attendre un événement X ou un autre événement comme l'expiration d'un timer. Lorsque l'événement attendu se produit, *la boucle d'événement* exécute un code C ou Tcl pour répondre à l'événement. Lorsqu'on a complètement répondu à l'événement, le contrôle retourne à la *boucle d'événement* pour attendre le prochain événement intéressant. Un nouvel événement qui arrive est ignoré si on est en train de prendre en charge l'événement courant.
Signalons que les OICs text et canvas supportent un binding interne ; Pour cela on associe des tags au texte ou aux items graphiques et on applique un binding aux tags. Un tag est tout simplement un string qui est utilisé pour identifier l'item graphique dans le cas d'un OIC canvas ou une zone de texte dans le cas d'un OIC text

2.3.6. Les mécanismes de sélection

Ce sont les mécanismes qui permettent l'échange d'information entre les OICs et l'application. Un utilisateur peut sélectionner un ou plusieurs objets dans un OIC, par exemple

sélectionner une portion du texte. Une fois la sélection faite, l'utilisateur peut invoquer une commande dans un autre OIC pour extraire l'information sélectionnée.
Tk utilise la commande *selection* pour sélectionner ou extraire la sélection. Lorsqu'on extrait la sélection, on peut demander plusieurs sortes d'informations qui sont référencées par des *targets*. Le *target* le plus utilisé est le STRING. Par exemple si la sélection d'un texte est extraite avec le target STRING, c'est le contenu du texte sélectionner qui est renvoyé. C'est la commande *selection get* qui permet d'extraire la sélection. La commande *selection own* permet de déterminer si un OIC a la sélection dans une application. La commande *selection clear* permet d'annuler la sélection.
Les OICs standards tels que entry, text possèdent déjà un code C qui leur implémente les mécanismes de sélection. Il est aussi possible d'écrire un script Tcl qui implémente la sélection mais cela n'est pas nécessaire dans le cadre de ce travail.

2.3.7. Le focus d'entré

A un moment donné, une seule fenêtre de l'application a le focus d'entrée. Toutes les touches du clavier reçues par l'application seront dirigées vers la fenêtre ayant le focus et les événement associés pourront être déclenchés.
Il existe deux modèles pour gérer le focus d'entrée : le modèle implicite du focus et le modèle explicite. Dans le modèle implicite le focus suit les déplacements de la souri. Les touches du clavier sont dirigées vers la fenêtre sous le pointeur de la souri et la fenêtre ayant le focus change implicitement lorsque la souri se déplace d'une fenêtre à l'autre. Dans le modèle explicite, le focus est affecté explicitement, les déplacement de la souri ne change pas le focus.
Tk implément le modèle explicite. C'est la commande focus qui permet d'affecter un focus. Elle reçoit en argument le nom de l'OIC qui aura le focus.

2.3.8. Les interactions modales

Les mécanismes d'interactions permettent de limiter les fenêtres avec lesquelles l'utilisateur peut interagir. Le plus simple exemple de l'interaction modale est la boîte de dialogue : par exemple pour enregistrer un fichier, l'application peut obliger l'utilisateur à fournir le nom du fichier avant d'enregistrer.
Tk fournit deux mécanismes pour l'interaction modale. Le premier est fourni par la commande *grab* qui permet de restreindre l'utilisateur de telle façon qu'il ne peut interagir qu'avec un nombre limité de fenêtres de l'application. Le deuxième est fourni par la commande *tkwait* qui permet de suspendre l'exécution d'un script jusqu'à ce qu'un événement particulier se produit ; par exemple répondre à une boîte de dialogue.

2.3.9. Le gestionnaire de fenêtres

Pour chaque application qui utilise le système X window, il y a un processus spécial appelé gestionnaire de fenêtres. Sa fonction principale est de contrôler l'arrangement des fenêtres de plus haut niveau à l'écran. De ce point de vue, il ressemble au gestionnaire de placement sauf qu'au lieu de gérer l'arrangement des fenêtres internes d'une application, il gère les fenêtres de plus haut niveau des différentes application. Les gestionnaires de fenêtres permettent à chaque application d'avoir une taille et une position particulière pour ses fenêtres de plus haut niveau. Les gestionnaires des fenêtres remplissent aussi d'autres fonctions entre autre, elles permettent à des fenêtres d'être icônifiées ou déicônifiées, elles communiquent à l'application l'occurrence de certains événement, etc.
Les exemples des gestionnaires de fenêtres sont par exemple : *mwm,twm* et *tvtwm*. Nous avons utilisé le gestionnaire de fenêtres *mwm*.
C'est la commande *wm* qui permet de communiquer avec le gestionnaire des fenêtres. Cependant, même si on n'utilise pas la commande *wm*, Tk continue à communiquer avec le gestionnaire des fenêtres tant qu'une fenêtre de plus haut niveau de l'application se trouve à

l'écran. Chaque fenêtre de plus haut niveau apparaît avec sa taille naturelle, celle qui est obtenue avec les mécanismes associés au gestionnaires de placement. Pour change cette taille, cela est prise en charge par le gestionnaire de fenêtres. Par défaut, un utilisateur ne peut modifier d'une façon interactive la taille des fenêtres. Pour cela, il faut déterminer l'intervalle de valeurs admissibles avec les commandes *wm minsize* et/ou *wm maxsize*. L'unité de mesure utilisée est le *pixel*.
Pour contrôler la position des fenêtres, on utilise la commande *wm geométry*. Les valeurs positives sont mesurées à partir du coin supérieur gauche, les valeurs négatives à partir du coin inférieur droit.
La commande *wm iconify* permet d'icônifier la fenêtre, la fenêtre n'apparaît pas à l'écran et la commande *wm deiconify* a l'effet inverse.
La commande *wm withdraw* enlève la fenêtre de l'écran.
Le gestionnaire de fenêtres ajoute un frame décoratif aux fenêtres. La commande *wm title* permet de spécifier le titre qui sera affiché dans le frame décoratif.

2.4. Interface entre une application et l'interface graphique réalisée avec Tk

Pour impléménter un interface graphique au dessus d'une application existante, on peut utiliser un mécanisme de pipe. On lance le processus lié à l'application qui va écrire dans le pipe. Ceci peut être réalisé au moyen d'une procédure activée par un bouton de commande par exemple. Ensuite le processus lié à l'interface graphique lit les données dans le pipe et les affiche dans une fenêtre par exemple.

CHAPITRE 3 : INTERFACE GRAPHIQUE D'ASAX.

(Réf. 2, 3)

Nous sommes inspiré de l'interface graphique qui est présenté dans le manuel *User's Guide* d'ASAX et nous avons appliqué la méthode de conception des interfaces graphiques vue au cours d'IHM. Cette méthode se base sur l'analyse de la tâche et sur plusieurs critères ergonomiques dans le choix des objets interactifs et dans la présentation. Elle propose une démarche structurée pour concevoir une IHM. Nous verrons effectivement qu'un certain nombre de fenêtres de l'application dérivent directement de l'analyse de la tâche et que d'autres dérivent de la prise en compte des critères ergonomiques et de cohérence dans la présentation.

3.1. Description du cas

3.1.1. Enoncé de la tâche principale envisagée

L'objectif de l'interface graphique d'ASAX est d'offrir un environnement convivial à l'utilisateur. Par rapport à l'interface de commande, l'avantage est de permettre par exemple l'utilisation de plusieurs fenêtres en même temps, la création des scénarios directement à partir des modules, etc.

L'interface graphique devra supporter tous les aspects interactifs des fonctions ASAX :

- Créer et modifier des fichiers de description de données *(data description files ou DDF)*, des scénarios et des modules RUSSEL : l'interface permettra à l'utilisateur d'ouvrir des fichiers existants ou de créer de nouveaux fichiers. Pour cela, l'utilisateur disposera d'un éditeur fourni par l'interface pour traiter les fichiers de description de données et les modules. Il pourra aussi utiliser un autre éditeur qu'il choisira à sa guise. L'interface fournira des facilités pour créer les scénarios en insérant supprimant des modules.
- Permettre un check des scénarios et des modules RUSSEL.
- Exécuter l'analyse : l'exécution de l'analyse supposera la définition du mode d'analyse (qui peut être *standard, conversion ou mixte*) et l'utilisateur devra fournir le fichier de description de données et le scénario courants.
- Afficher les résultats du check et de l'analyse. Les résultats seront affichés à l'aide de l'éditeur et l'utilisateur aura la possibilité de les enregistrer s'il le souhaite.

L'interface graphique ne supportera pas les aspects *off-line* d'ASAX tels que la préparation des routines C et de l'adaptateur de format, la génération d'un nouvel exécutable d'ASAX.

Deux fenêtres seront toujours présentes à l'écran : la fenêtre de description des données et la fenêtre de scénario. A tout moment, il existe une et une seule instance de ces fenêtres. L'interface possède aussi une fenêtre qui sert à afficher les routines C intégrés dans l'application. Il faut noter que ces routines C peuvent être affichés mais pas modifiés. Toutes les fenêtres ont comme titre : *Identifiant de l'application : titre de la fenêtre*. L'identifiant de l'application est par défaut ASAX mais il peut être modifié par l'utilisateur au cours d'une session via la fenêtre réservée à cet effet. Le titre de la fenêtre est relatif à l'usage de fenêtre en question. Pour une fenêtre qui sert a éditer un fichier, le titre de la fenêtre sera le nom du fichier édité.

Les paramètres relatifs à la tâche dérivent de l'étude du cas. Ils sont donnés ci-dessous :

- Pré-requis : modérés à maximaux. Les pré-requis d'une tâche expriment les connaissances du système existant/futur que l'utilisateur doit posséder en vue de l'utilisation efficace du système. L'utilisation du système suppose qu'on a une bonne connaissance du domaine d'application.
- Productivité : faible. La productivité d'une tâche représente le nombre d'exécution moyen de la tâche par unité de temps. Excepté le cas où l'on est en mode analyse temps réel, la productivité sera faible.

- Environnement objectif de la tâche : inexistant. L'environnement objectif de la tâche traduit la présence ou l'absence d'objets spécifiques manipulés dans le cadre de l'accomplissement de la tâche, indépendamment de tout système.
- Reproductivité de l'environnement : l'environnement objectif est inexistant donc, inutile de parlé de reproductivité.
- Structuration de la tâche : faible. La structuration de la tâche explique le degré de liberté ou de contrainte que l'utilisateur a dans son accomplissement.
- Importance de la tâche : élevé. L'importance de la tâche informe quant au caractère fondamental, crucial, vital d'une tâche. Si on considère le point de vue sécurité, on ne peut douter du caractère fondamental de la tâche.
- Complexité cognitive de la tâche : élevée. La complexité de la tâche manifeste le degré de complexité cognitif, manipulatoire, intellectuel quant aux sous-tâches et actions mises en jeu en vue d'accomplir la tâche. Ici la complexité cognitive est élevée vue le caractère technique des informations manipulées.

3.1.2. La description des stéréotypes d'utilisateur

ASAX est destiné à trois types d'utilisateurs :

- les administrateurs de système qui sont responsables de l'installation et de la génération de l'exécutable d'ASAX.
- Les experts qui peuvent du système qui connaissent bien les langages C et RUSSEL.
- Les utilisateurs ordinaires qui utilisent les programmes écrits par les experts pour faire l'analyse. Ces utilisateurs doivent néanmoins avoir une bonne connaissance du domaine d'application.

On peut donc considérer un seul stéréotype d'utilisateurs pour une utilisation efficace du système : les utilisateurs expérimentés.

Les paramètres sont les suivants :

- Expérience de la tâche : riche, car l'utilisateur connaît bien la tâche et les sous-tâches de l 'application.
- Expérience de systèmes informatiques : riche, car le niveau de connaissance de l'utilisateur en matière d'utilisation des systèmes informatiques est celui d'un expert, que ce soit dans le domaine de la tâche ou non.
- Motivation : élevée, car l'utilisateur maîtrise bien la tâche et le système.
- Expérience d'un moyen d'interaction complexe : riche, car l'utilisateur est expert des systèmes informatiques. En plus, la plate-forme considérée pour ASAX est Unix qui est un système d'interaction complexe.

3.1.3. Description de l'environnement

Les paramètres décrivant le poste de travail de l'utilisateur sont :

- type de traitement : multitraitement ; il est possible de lancer l'analyse et continuer à s'occuper d'autres tâches ;
- capacité de traitement : la capacité de traitement revient à la répétitivité de la tâche. Celle-ci est faible.

3.2. Analyse de la tâche

Nous utiliserons la méthode TKS (Task Knowlegde Structures) vue au cours d'IHM.

3.2.1. Décomposition de la tâche principale d'analyse

En partant de la description des fonctionnalités que doit remplir l'interface graphique, nous pouvons dresser une décomposition de la tâche comme suit :

Décomposition en but et sous-buts

1. **Analyser un fichier**
   - 1.1.Définir un ddf
     - 1.1.1. Créer un nouveau ddf
     - 1.1.2. Ouvrir un ddf existant
   - 1.2.Définir un scénario
     - 1.2.1. Créer un nouveau scénario
       - 1.2.1.1.Déterminer les modules
         - 1.2.1.1.1. Ecrire un nouveau module
         - 1.2.1.1.2. Choisir les modules existants (ajouter ou supprimer les modules)
     - 1.2.2. Ouvrir un scénario existant
       - 1.2.2.1.Déterminer les modules
         - 1.2.2.1.1. Ecrire un nouveau module
         - 1.2.2.1.2. Choisir les modules existants (ajouter ou supprimer)
   - 1.3.Analyser
     - 1.3.1. Déterminer le mode d'analyse
     - 1.3.2. Lancer l'analyse

Diagramme des buts et sous-buts

Le diagramme ci-dessous représente la décomposition de la tâche en buts et sous-buts. Les flèches en trait plein montrent les relations de structuration. Les flèches en trait pointillé montrent les relations de séquencement.

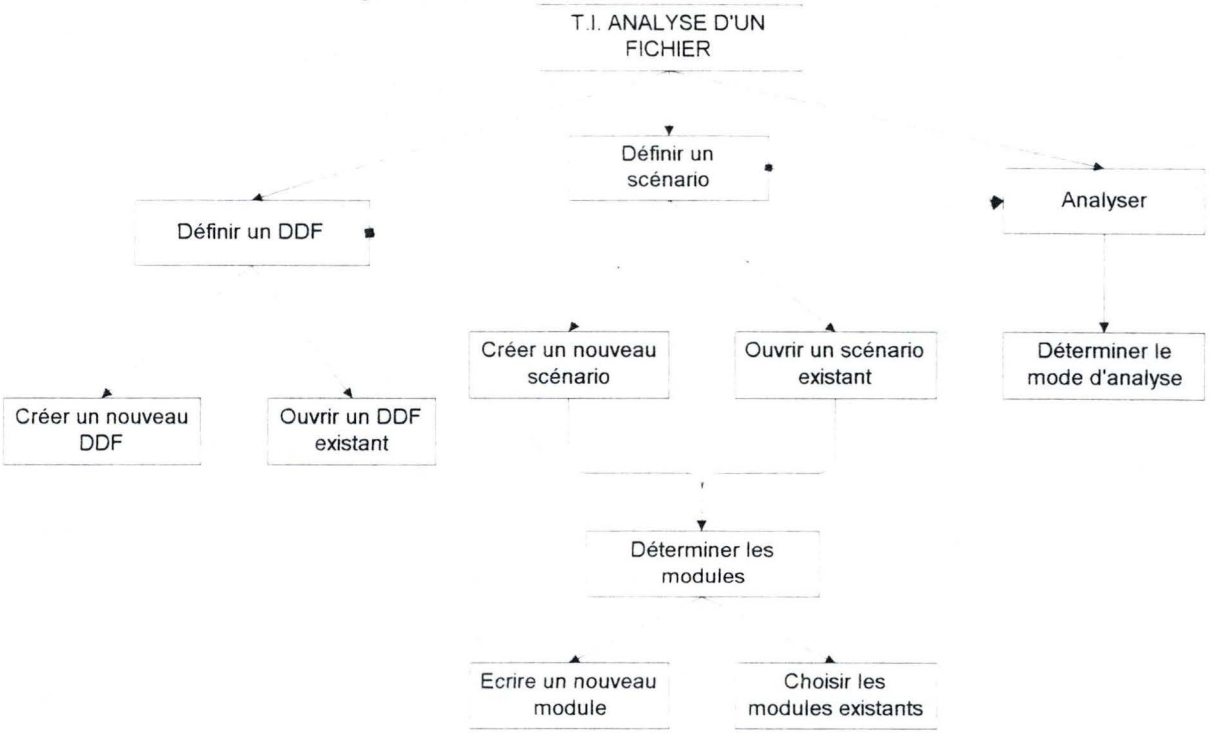


Figure 3.1 Décomposition de la tâche principale en buts et sous-buts

Décomposition en procédures
A chaque sous-buts correspond une sous-tâche et donc la tâche d'analyse comporte trois sous-tâches : définir un ddf, définir un scénario, analyser.
La décomposition en procédure de la première sous-tâche est la suivante :

Le langage utilisé comprend uniquement des structures du type :

SI *condition* **ALORS** *action* **FINSI**

Légende :
→ message : message en entrée
← message : message en sortie

Définir un ddf
ddf := **Faux**
choix := selection (menu)
SI choix = nouveau_ddf
    ALORS
        creer_ddf (→NewDdf,←Ddf)
        ddf := **Vrai**
FINSI
SI choix = ddf_existant
    ALORS
        ouvrir (→DdfFileName, ←Ddfile)
        ddf := **Vrai**
FINSI

La décomposition en procédure de la deuxième sous-tâche est la suivante :

Définir un scénario

```
scenario := Faux
choix := sélectionner (menu)
choix_mdl := sélectionner (ModuleFileNameLste)
action := delte_mdl, add_mdl, check.
SI choix = nouveau_sce
        ALORS
                creer_sce (→ NewSce,←NewScefile)
                scenario := Vrai
        SI choix = nouveau_mdl
                ALORS
                        ecrire_mdl (→NewMdl,←NewMdlfile)
        FINSI
        SI action = add_mdl
                                ALORS
                                        MdlFile := recherche_MdlFileName
                                        ajouter (→MdlFile, →NewScefile)
                        FINSI
        SI choix_mdl = MdlFileName
                ALORS
                        SI action = delete_mdl
                                ALORS
                                        effacer (→MdlFileName)
                        FINSI
                        SI action = check
                                ALORS
                                        check_mdl (→MdlFileName,←ReportCheck)
                        FINSI
        FINSI
        scenario := Vrai
        SI action =check
                ALORS
                        Check_sce (→Sce, ←ReportCheck)
FINSI
FINSI
SI choix = sce_existant
        ALORS
        ouvrir (→sce_file, ←ModuleListe)
        scenario := Vrai
        SI action = add_mdl
                ALORS
                        MdlFile := recherche_MdlFileName
                        ajouter (→MdlFile)
        FINSI
        SI choix_mdl = MdlFileName
                ALORS
                        SI action = delete_mdl
                                ALORS
                                        effacer (→MdlFileName)
```

FINSI
SI action = check_mdl
ALORS
check (→MdlFileName,←ReportCheck)
FINSI
FINSI
SI action =check
ALORS
Check_sce (→Sce, ←ReportCheck)
FINSI
FINSI
La décomposition en procédure de la troisième sous-tâche est la suivante :
Analyser
choix := sélectionner (menu)
SI choix = analyse
ALORS
analyser (→Ddf,→Sce,→ModeAnalyse ,←ReportAnalyse)
FINSI

3.3. Expression du produit d'analyse de la tâche

3.3.1. Schéma entité association

Connaissant les différents éléments de notre application, nous pouvons donc faire le schéma entité association correspondant :

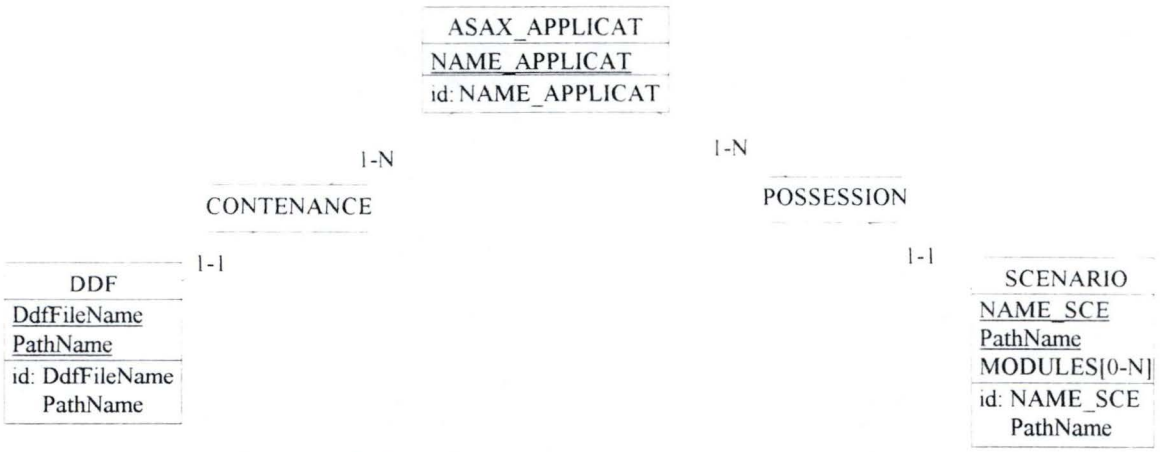
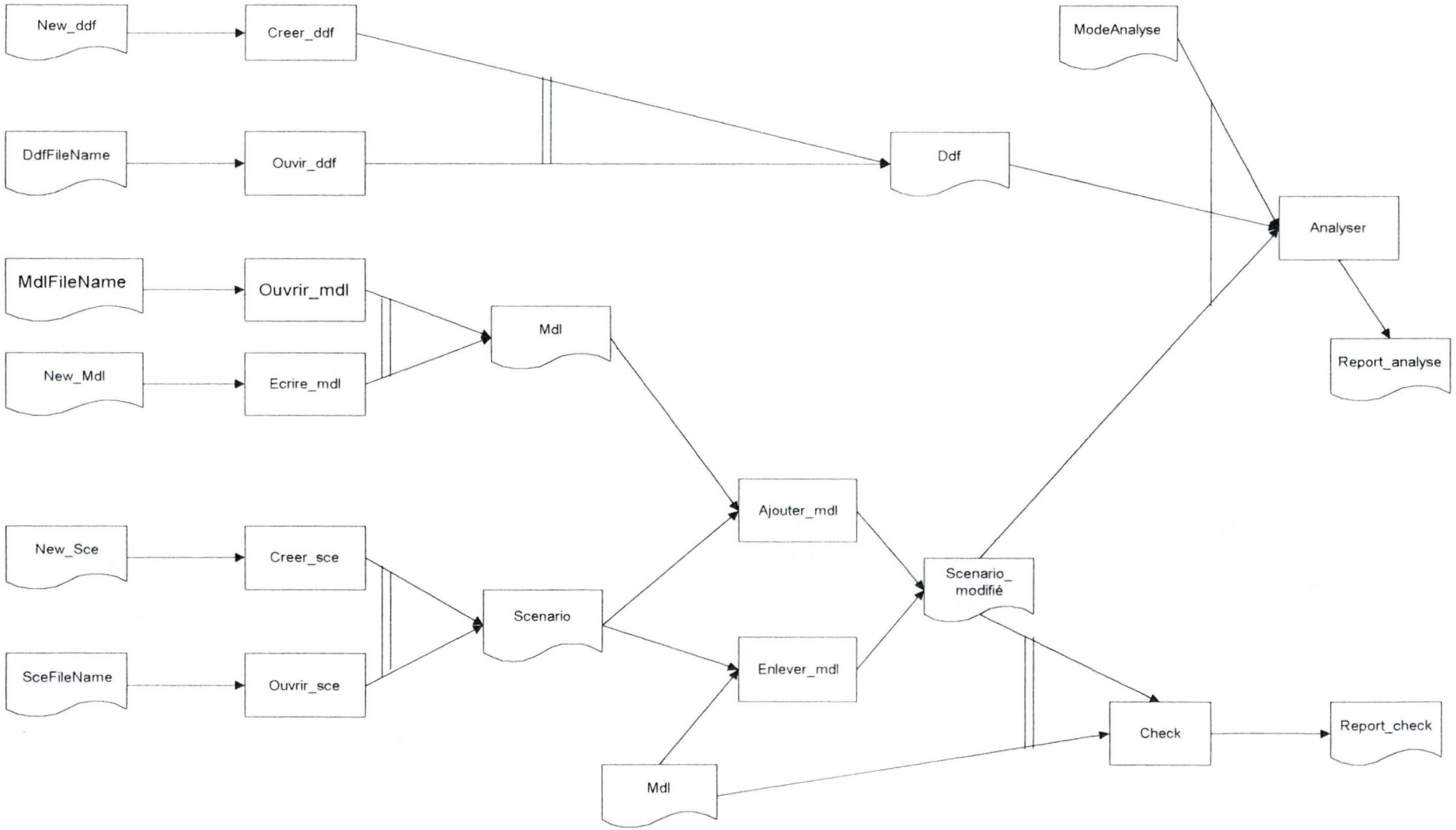


Figure 3.2 Schéma entité association de l'application.

Ce schéma traduit le fait que l'application ASAX est identifiée par le nom de l'application. ASAX contient 1 à N fichier de description de données et 1 à N scénarios.
Un fichier de description de données est identifié par le nom du fichier est son chemin d'accès et il est associé à une et une seule application ASAX.
Un scénario est identifié par le nom du fichier de scénario et son chemin d'accès. Il contient 0 à N modules et est associé à une et une seule application ASAX
La cardinalité [0-N] de l'attribut MODULES tient compte du fait que lors de la création d'un nouveau scénario, la liste des modules est vide.

Les fonctions importantes pour notre application sont donc les suivantes :
creer_ddf (→NewDdf,←Ddfile)
ouvrir (→DdfFileName, ←Ddfile)
creer_sce (→ NewSce,←NewScefile)
ecrire_mdl (→NewMdl,←NewMdlfile)
ajouter (→MdlFile)
effacer (→MdlFileName)
check (→MdlFileName,←ReportCheck)
check (→Sce,←ReportCheck)
analyser (→Ddf,→Sce,→ModeAnalyse ,←ReportAnalyse)

3.3.2. Le graphe d'enchaînement des fonctions



Légende :

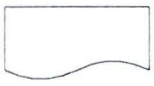
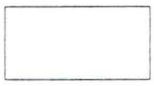
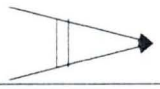
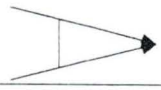
			
Message	Fonction	OU exclusif	ET

Figure 3.3 Graphe d'enchaînement des fonctions

Dans le graphe ci-dessus, nous avons omis la fonction « enregistrer » pour pouvoir le rendre plus lisible, elle est intuitive.

3.3.3. Choix des attributs du dialogue

Les quatre attributs caractérisant le dialogue de l'interface sont :

1. Le contrôle du dialogue : il est globalement externe pour la tâche, c'est à dire à l'initiative de l'utilisateur. Cependant, il est partiellement interne pour le menu principal qui permet de lancer l'application : ouvrir un ddf, ouvrir un scénario, ouvrir un module, ... Ce menu engendre un contrôle interne puisqu'il est proposé et contrôlé par l'application.
2. Le mode de dialogue : il est asynchrone entre les quatre sous-tâches : l'ordre d'exécution des actions est non-prédéterminé, non-séquentiel. Parfois, au sein d'une sous-tâche le mode de dialogue devient synchrone pour éviter les erreurs de l'utilisateur.
3. Le mode de déclenchement des fonctions : il est manuel explicite est affiché car les fonctions doivent être déclenchées à l'initiative de l'utilisateur à l'aide des actions prévues à cet effet.
4. La métaphore : elle est basée sur le mini-monde. La métaphore est celle de la confrontation entre l'utilisateur et le système d'information. L'interface est un monde dans lequel l'utilisateur peut agir et ce monde change d'état en fonction des actions de l'utilisateur. L'utilisateur agit en tant qu'acteur et non en tant que donneur d'ordre à un robot.

En résumé, le contrôle du dialogue est externe, le mode de dialogue, asynchrone ; le mode de déclenchement est explicite et affiché ; la métaphore est celle du mini-monde.
En fonction des paramètres que nous avons identifiés dans les sections précédentes, nous constatons en parcourant les tableaux de dérivation des styles d'interaction que nous pouvons retenir les styles d'interaction suivants : la sélection de menu, le multi-fenêtrage, le remplissage de forme. Ces styles d'interaction n'ont pas été choisis par application pure et simple des correspondances trouvées dans les tableaux, puisque ces derniers servent de base pour la réflexion du concepteur. On a essayé aussi de suivre la conception donnée dans le manuel *User Guide* d'ASAX.

3.4. Définition de la présentation

3.4.1. Identification des unités de présentation(UP)

Les unités de présentation peuvent être dérivées directement du graphe d'enchaînement des fonctions. A chaque sous-tâche correspond une unité de présentation et donc nous aurons trois unités de présentation que nous baptiserons :

1. UP1 : Déterminer un DDF
2. UP2 : Déterminer un Scénario
3. UP3 : Analyser.

Cela est représenté dans le graphe donné ci-dessous :

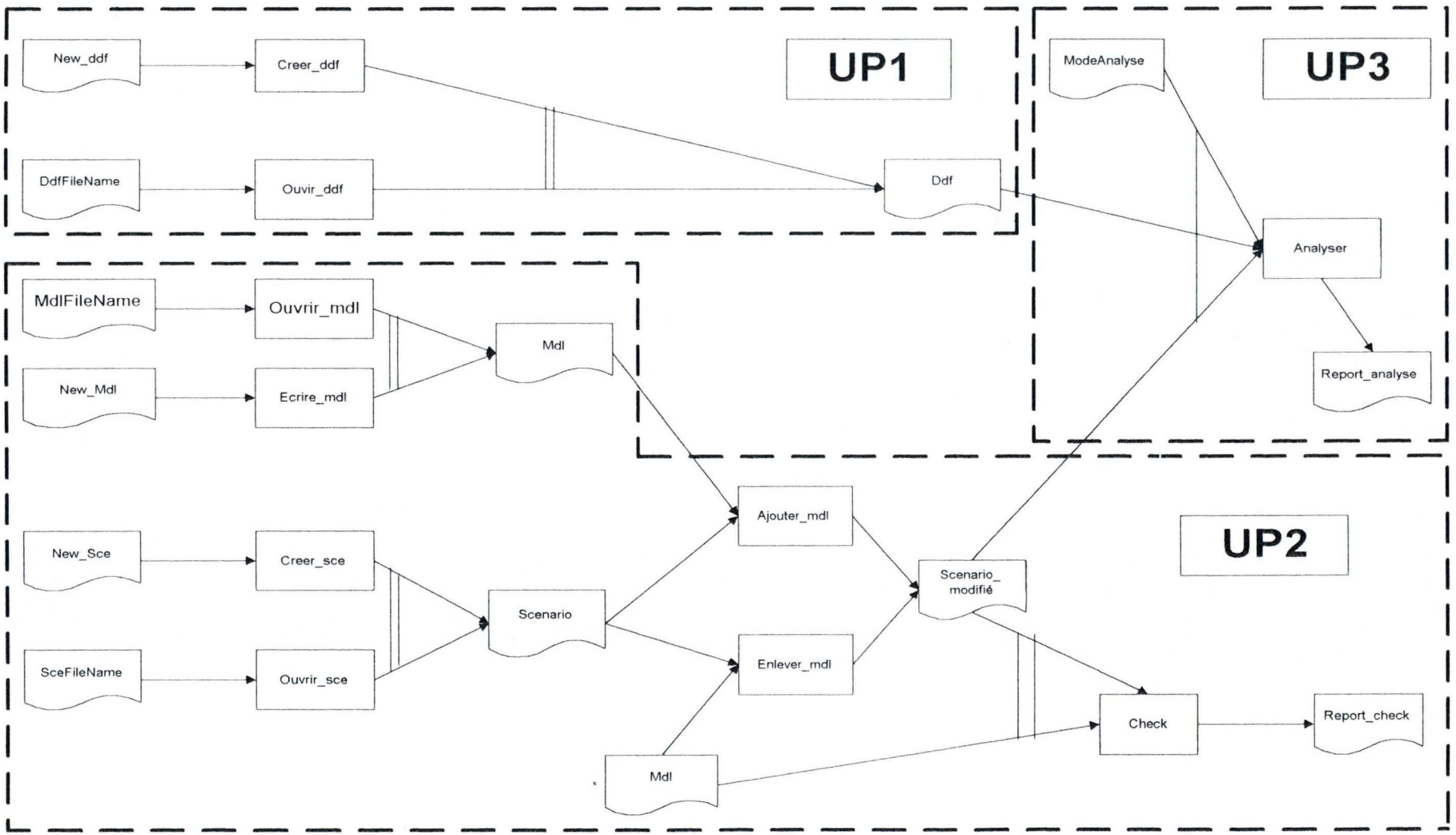


Figure 3.4 Définition des unités de présentation.

3.4.2. Identification des fenêtres

L'identification des fenêtres revient à décomposer chaque unité de présentation en sous-graphes qui correspondent chacun à une fenêtre. Il existe différents critères d'identification traduisant une décomposition plus ou moins fine de l'UP en fonction du niveau de l'utilisateur envisagé. Plus la décomposition est fine et plus l'interface convient à l'utilisateur novice.
Les différents critères d'identification sont :

- L'identification minimale : une information par fenêtre.
- L'identification maximale : une fenêtre par UP.
- L'identification entrées/sorties : toutes les informations externes sont groupées dans une fenêtre et toutes les informations externes en sortie dans une autre fenêtre.
- L'identification fonctionnelle : toutes les informations en E/S sont groupées dans une seule et même fenêtre.
- L'identification libre : on défini chaque fenêtre comme on veut.
- L'identification groupée : à partir des identifications précédentes, on peut décider de grouper plusieurs fenêtres ensembles pour des raison d'ergonomie pour autant que la charge de travail n'excède pas les limites cognitives de l'utilisateur.

UP1 : Déterminer un DDF et UP3 : Analyser
On opte pour l'identification entrées/sorties. On aura une fenêtre en fonction de l'action choisie et le résultat se trouvera dans une autre fenêtre.
UP2 : Déterminer un Scénario
Ici, on opte pour l'identification entrées/sorties et une identification fonctionnelle.

Notes : L'action enregistrer que nous avons omise du graphe d'enchaînement donnera lieu lui aussi à une fenêtre.

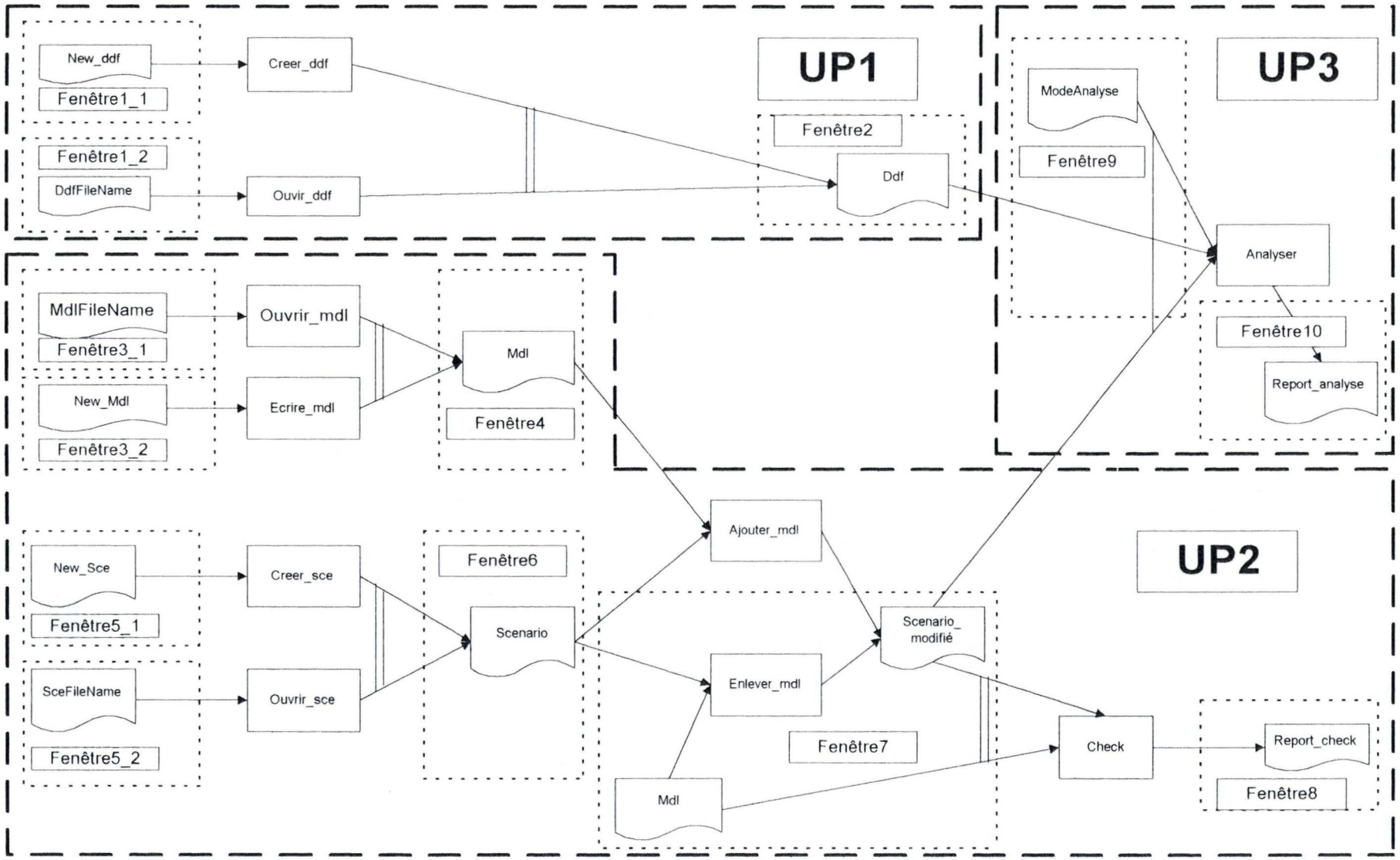


Figure 3.5 Définition des fenêtres logiques de l'application

3.4.3. Sélection des objets interactifs abstraits (OIA)

Les fenêtres qui viennent d'être identifiées sont des fenêtres logiques. Chaque fenêtre logique contient des informations qui doivent être saisies, affichées, informations de commandes, etc. Ces informations seront présentées par des OIA. Ces derniers sont indépendants des environnements physiques contrairement aux objets interactifs concrets qui peuvent changer d'appellation suivant l'environnement physique. Pour le choix des OIA, nous appliquerons les règles ergonomiques du corpus minimal des applications de gestion.
En vertu de ces règles ergonomiques, les OIA associés aux informations à afficher ou à saisir sont :

- Les noms de fichiers seront choisis à l'aide d'une liste de sélection de fichier (LSF)
- Le contenu d'un fichier sera affiché ou saisi à l'aide d'une fenêtre d'édition textuelle (FED)
- Le nom d'un fichier sera saisi ou affiché à l'aide d'un champ d'édition unilinéaire (EDU)
- Le type d'action à effectuer et le mode d'analyse seront choisis à l'aide de bouton radio (BTR)
- Le noms de fichiers seront afficher à l'aide d'une liste de sélection (LSB).

3.4.4. Transformation des objets interactifs abstraits (OIA) en objets interactifs concrets (OIC)

A chaque OIA présenté ci-dessus nous faisons correspondre des OIC propres à l'environnement Tk :

Liste de sélection de fichier (LSF)	à programmer
Fenêtre d'édition textuelle (FED)	text
Champs d'édition unilinéaire (EDU)	entry
Bouton radio (BTR)	radiobutton
Liste de sélection (LSB)	listbox

3.4.5. Placement des OIC

Dans la suite nous allons présenter l'ensemble des fenêtres telles qu'elles ont été réalisées avec Tk. La description originale a été quelque peu modifiée en fonction du résultat de l'analyse de la tâche. Nous reprendrons chaque fois la description de la fenêtre avant de la présenter.

3.5. Les fenêtres de l'application

3.5.1. Les fenêtres déduites de l'analyse de la tâche

a) Les fenêtres

- La fenêtre de description de données : durant l'exécution d'ASAX, il existe une et une seule fenêtre de description de données à l'écran ; cette fenêtre affiche le fichier de description de données courant (DDF). Les fichiers DDF ont pour extension « *.ddf* ». L'édition de ce fichier est utile par exemple par exemple pour visualiser la définition des champs durant la création des modules. Le titre de cette fenêtre est le nom du fichier DDF. Ce fichier doit être directement affiché à l'écran et toute modification n'est prise en compte qu'après l'enregistrement. Si aucun fichier DDF n'est défini, la fenêtre affiche un message « *Data description file not defined* » pour inviter l'utilisateur à déterminer le fichier à analyser.

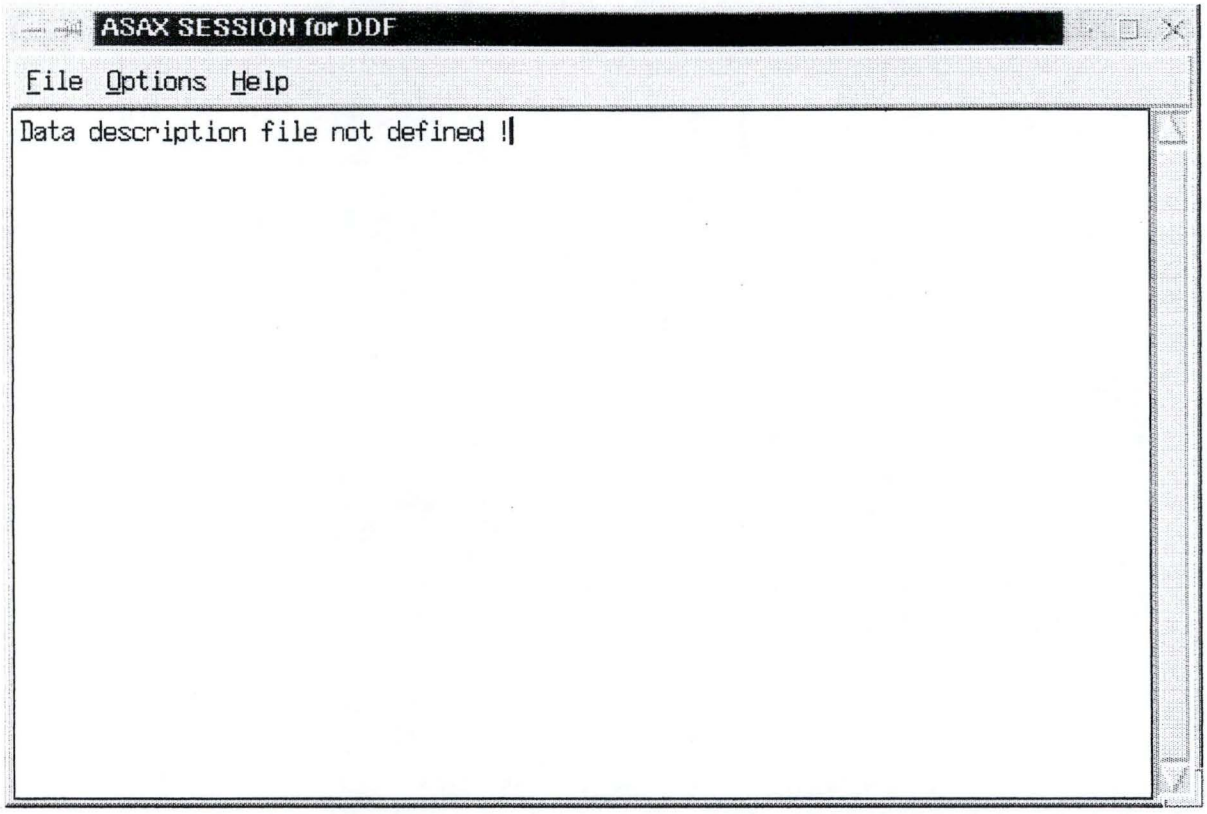


Figure 3.6 Fenêtre de description de données.

- La fenêtre de scénario : durant l'exécution d'ASAX, il existe une et une seule fenêtre de scénario à l'écran ; cette fenêtre affiche le scénario courant c'est à dire la liste des modules contenus dans un scénario (on affiche uniquement les noms des modules sans leur chemin d'accès). Cette liste est vide dans le cas de la création d'un nouveau scénario. Le fichier d'un scénario a pour extension « *.sce* ». Le titre de la fenêtre est le nom du scénario. On peut modifier un scénario via « *Options/Add Module* » et « *Options/Delete Module* » du menu principal. Un double-clique sur le nom du module permet d'ouvrir la fenêtre contenant la description de ce module. Si aucun scénario n'est ouvert, la fenêtre affiche un message « *Scenario not defined* » pour inviter l'utilisateur à définir un scénario.

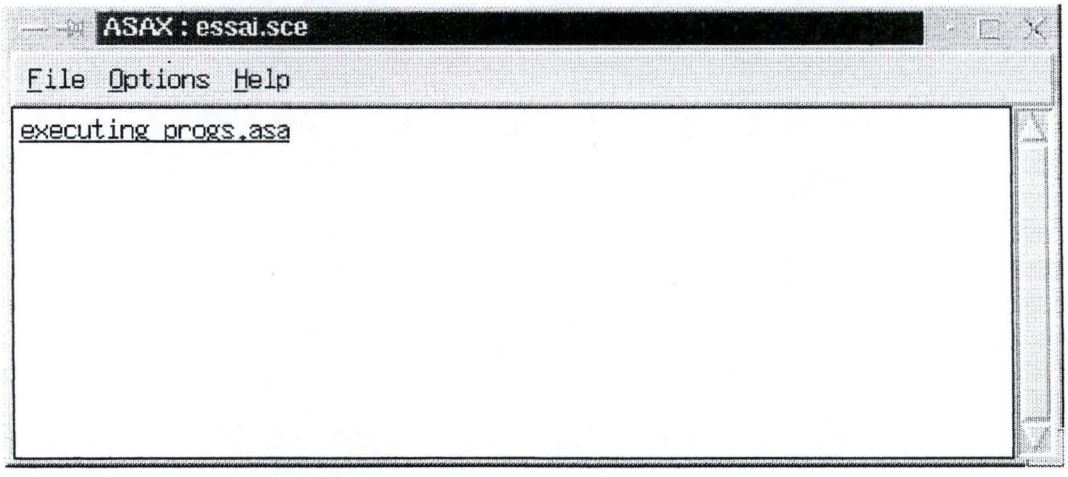


Figure 3.7 Fenêtre de scénario

- La fenêtre de module : cette fenêtre permet d'afficher l'édition d'un module RUSSEL. Cette fenêtre apparaît lorsque l'utilisateur désire créer un module via « *File/new* » du menu principal, ouvre un module existant via « *File/Open* » du menu principal, ou double-clique sur un nom de module dans la fenêtre de scénario. Le fichier d'un module a pour extension « *.asa* ». Le titre de la fenêtre est le nom du module. Toute modification n'est prise en compte par ASAX qu'après l'enregistrement.

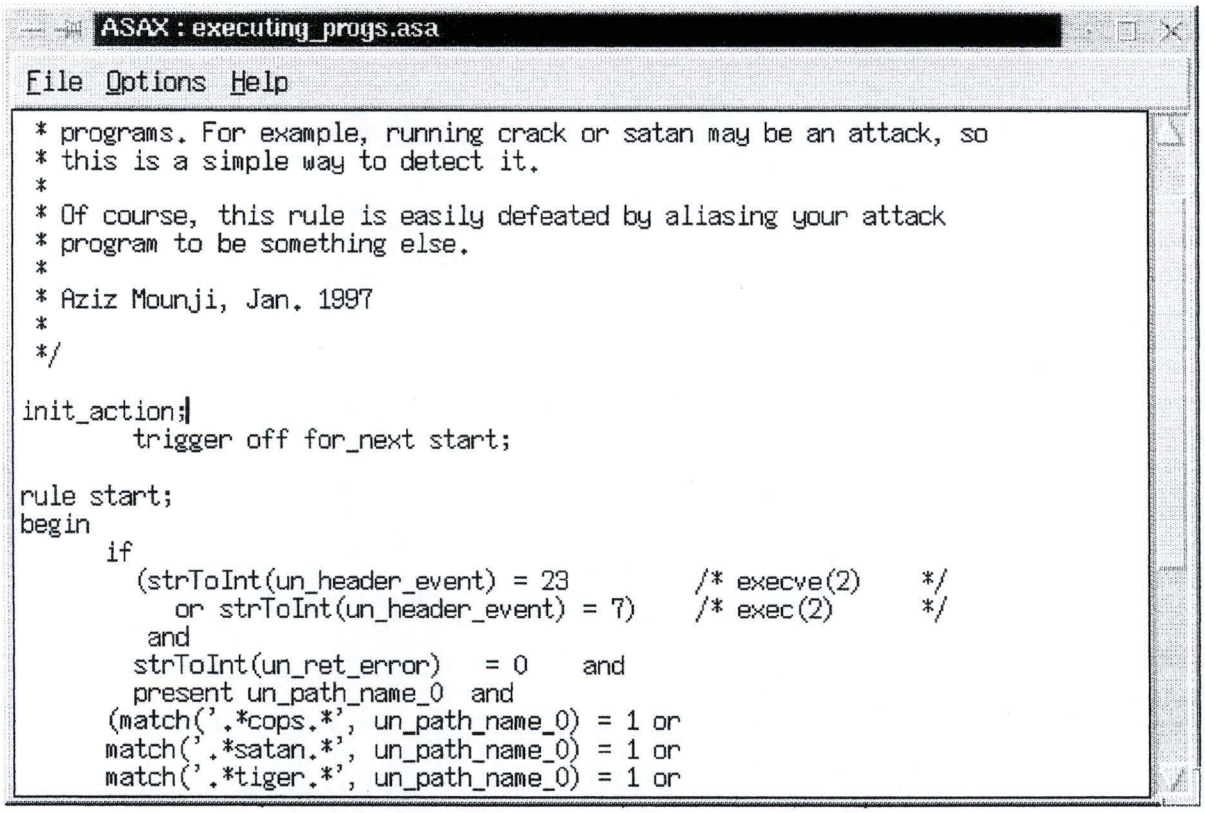


Figure 3.8 Fenêtre de module

- La fenêtre du résultat du check : cette lorsque l'utilisateur désire effectuer un check via « *Options/Check* » du menu principal lorsque la fenêtre de scénario ou de module est active. Un check global est réalisé lorsqu'il s'agit d'un scénario et un check local est effectué pour un module. Le but est d'afficher les erreurs trouvées lors du check. Le titre de cette fenêtre est le nom de l'erreur courant. Les erreurs trouvées sont perdues si elles ne sont pas enregistrées via « *File/Save* » du menu principal. *Un message d'avertissement apparaît à la fermeture de cette fenêtre pour demander à l'utilisateur s'il veut enregistrer le résultat.*

- La fenêtre du résultat d'analyse : cette fenêtre apparaît l'analyse est exécutée. Le but est d'afficher le résultat de l'analyse. Ce résultat est perdu s'il n'y pas enregistrement.

b) Les boîtes de dialogue

Les boîtes de dialogue sont utilisées pour échanger l'information entre l'utilisateur et l'application. Lorsqu'il n'est pas possible de continuer d'utiliser l'application sans avoir répondu à la boîte de dialogue, celle-ci est dite modale. Toute boîte de dialogue contient au moins deux boutons de commande : un pour l'enlever de l'écran, l'autre pour obtenir

l'information d'aide. Quelques boutons de commande spéciaux sont souvent rencontrés : Ok/Reset/Cancel/Help/Update.
Le bouton « Ok » : ce bouton sert à fermer la boîte de dialogue. Toutes les informations sont prises en compte et l'exécution se poursuit normalement. Parfois, ce bouton ne peut être sélectionné ; ceci survient lorsque l'activation de ce bouton provoquera une erreur. Si le bouton Ok peut être sélectionné, c'est le bouton par défaut sinon c'est le bouton Help.
Le bouton « Reset » : ce bouton annule toutes les informations qui sont affichées dans la boîte de dialogue à l'ouverture.
Le bouton « Cancel » : ce bouton ferme l boîte de dialogue ; toutes les informations qu'elle contient ne sont pas prises en compte. *L'application retourne à l'étape précédente l'ouverture de la boîte de dialogue*.
Le bouton « Help » : le bouton d'aide donne les informations d'aide on-line sur la boîte de dialogue comme la description des champs ou pourquoi le bouton Ok ne peut être sélectionné si c'est le cas.
Le bouton « Update » : ce bouton permet de mettre à jour les informations dans la boîte de dialogue sans la fermer.

c) La barre de menu d'ASAX

Nous allons décrire la barre de menu d'ASAX ainsi que les diverses boîtes de dialogue. Tous les items du menu ne sont pas toujours disponibles, ça dépend de la fenêtre active et de l'état courant de l'application.

- Le menu « File » :
  Ce menu contient les éléments suivants :
  - *New...* : Cet élément du menu permet de créer un nouveau DDF, un nouveau scénario ou un nouveau module. Lorsqu'il est sélectionné, une boîte de dialogue apparaît pour spécifier quel type de fichier sera créé. Cette boîte de dialogue est modale. Les radio buttons permettent de sélectionner quel fichier à créer (DDF, Scénario ou module). Le bouton Ok peut toujours être sélectionné.

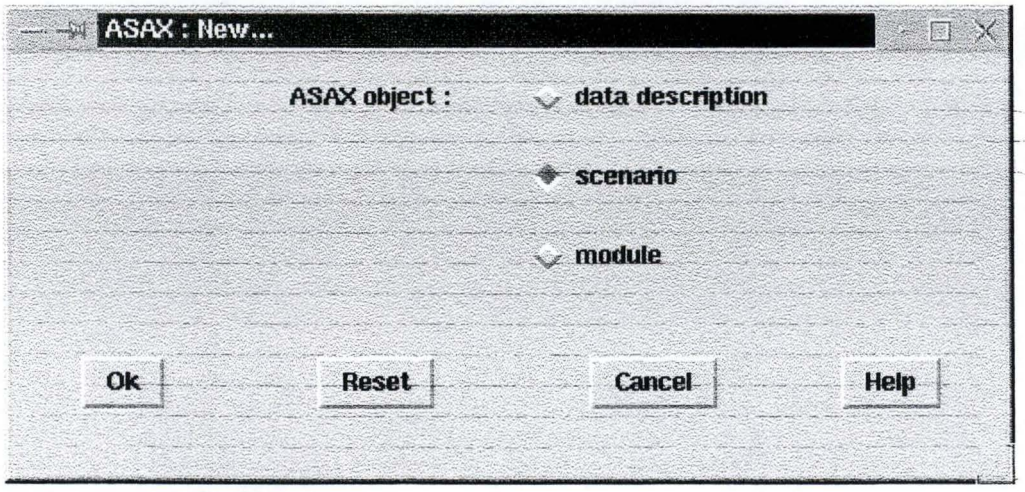


Figure 3.9 Boîte de dialogue New...

  - *Open...* : cet élément du menu permet d'ouvrir un fichier existant (DDF, Scénario ou module). Lorsqu'il est sélectionné, la boîte de dialogue *« Open... »* apparaît pour sélectionner le fichier désiré.

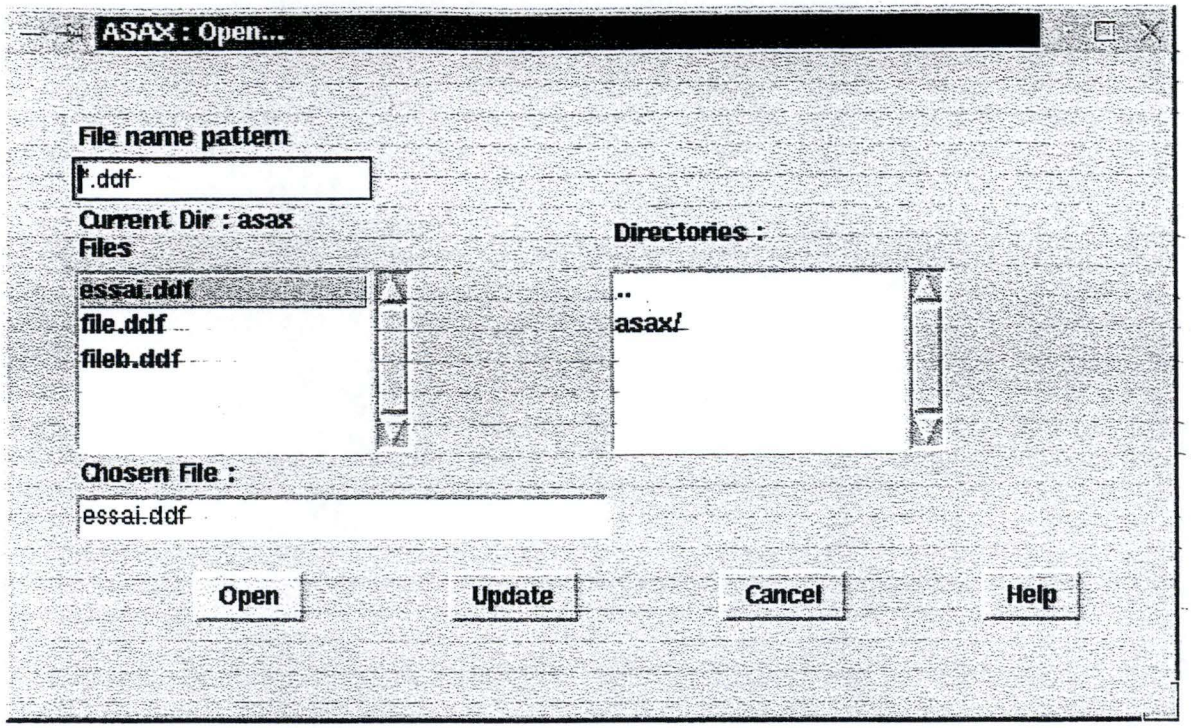


Figure 3.10 Boîte de dialogue Open...

- *Save* : cet élément du menu permet d'enregistrer le fichier courant.
- *Save as...* : il est identique au précédent sauf qu'il permet de sélectionner le nom du fichier à enregistrer.

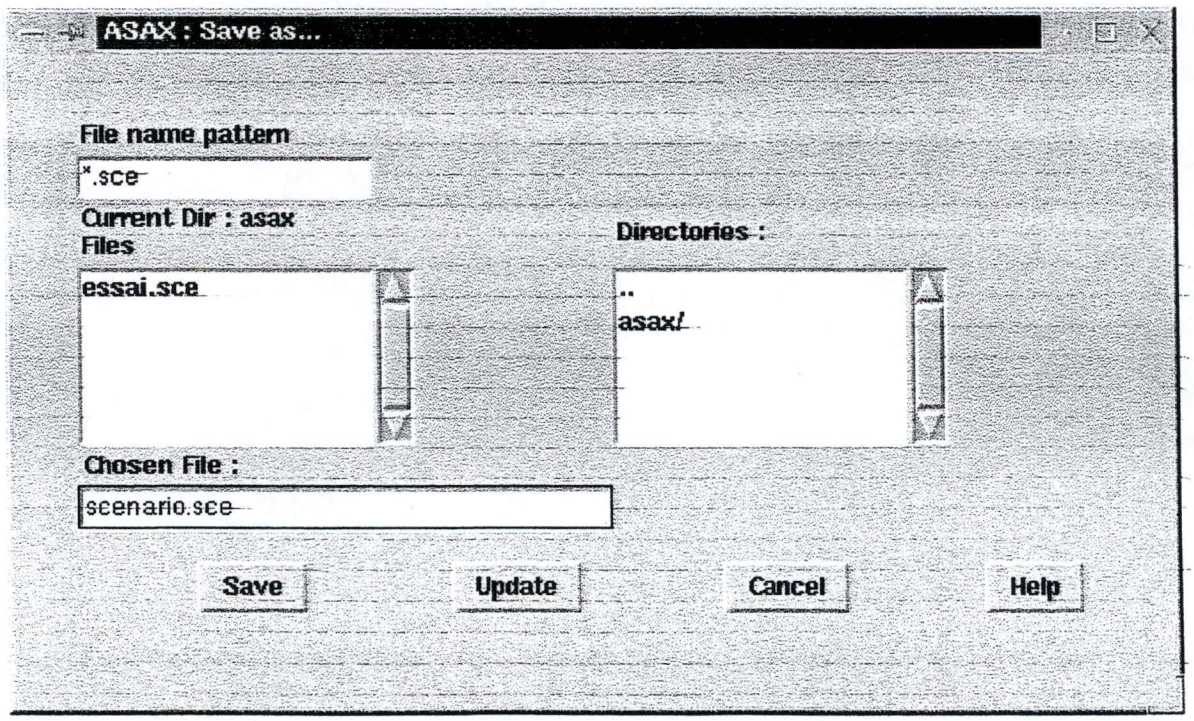


Figure 3.11 Boîte de dialogue Save...

- Le menu « Options » :
  Ce menu contient les éléments suivants :

♦ *Start Analysis* : cet élément permet de commencer l'analyse. C'est un menu en cascade qui contient à son tour trois options pour spécifier le mode d'analyse :

- *Standard mode...* : lorsqu'il est sélectionné, une boîte de dialogue apparaît permettant de spécifier le fichier NADF et le scénario à utiliser pour l'analyse.

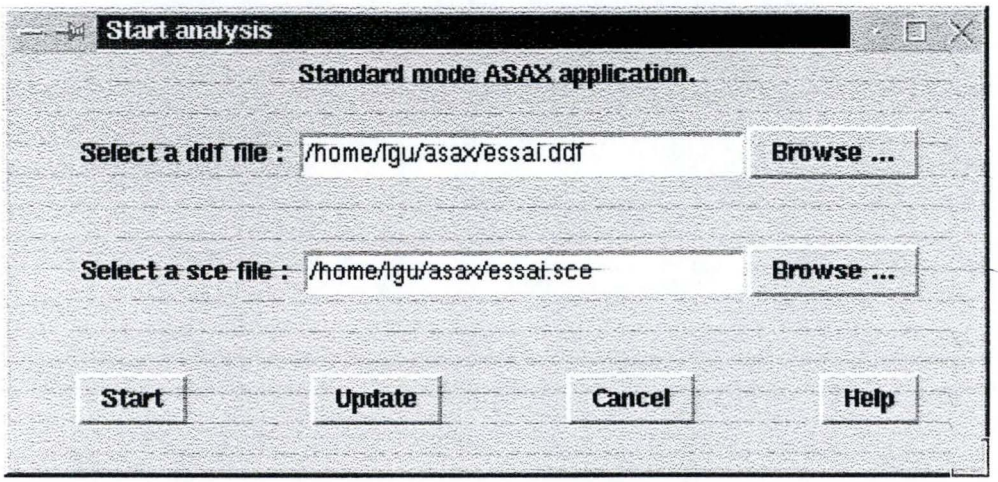


Figure 3.12 Boîte de dialogue Start analysis (standard mode)

- *Conversion mode...* : lorsqu'il est sélectionné, une boîte de dialogue apparaît semblable à la précédente.

- *Mixed-mode...* : lorsqu'il est sélectionné, une boîte de dialogue apparaît semblable à la précédente avec deux radio boutons permettant de spécifier quelle mode d'analyse sera utilisée (standard ou conversion).

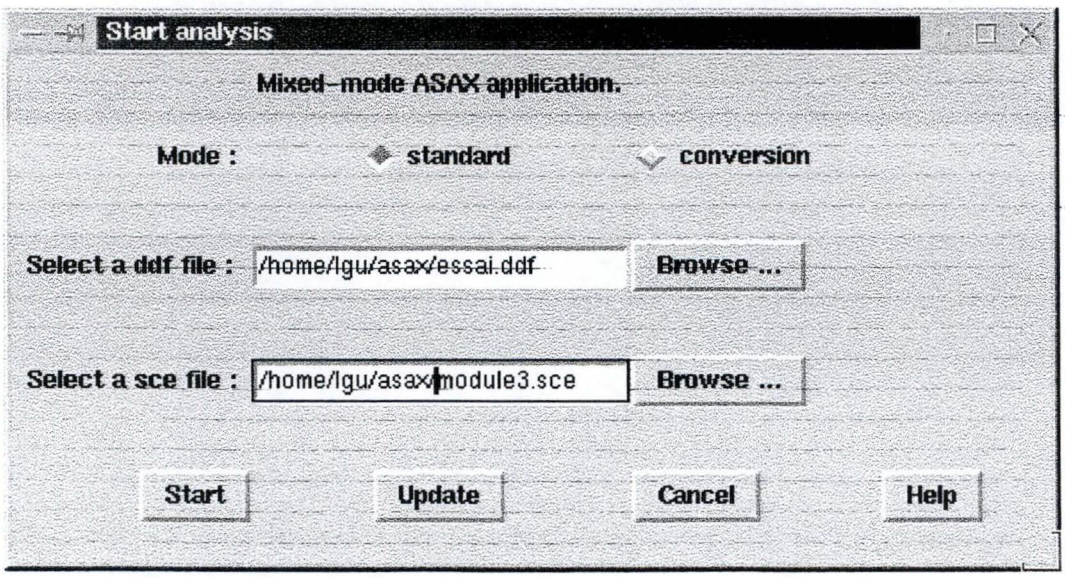


Figure 3.13 Boîte de dialogue Start analysis (mixed mode).

Lorsque l'analyse commence, une fenêtre de résultats apparaît qui affiche les résultats de l'analyse au fur et à mesure de leur apparition. *Il est possible d'effectuer plusieurs analyses en parallèle.*

♦ *Add Module...* : cet élément permet d'ajouter les modules existant au scénario courant. Il est disponible uniquement pour la fenêtre de scénario. Lorsqu'il est

sélectionné, une boîte de dialogue apparaît permettant de spécifier quel module doit être ajouté.

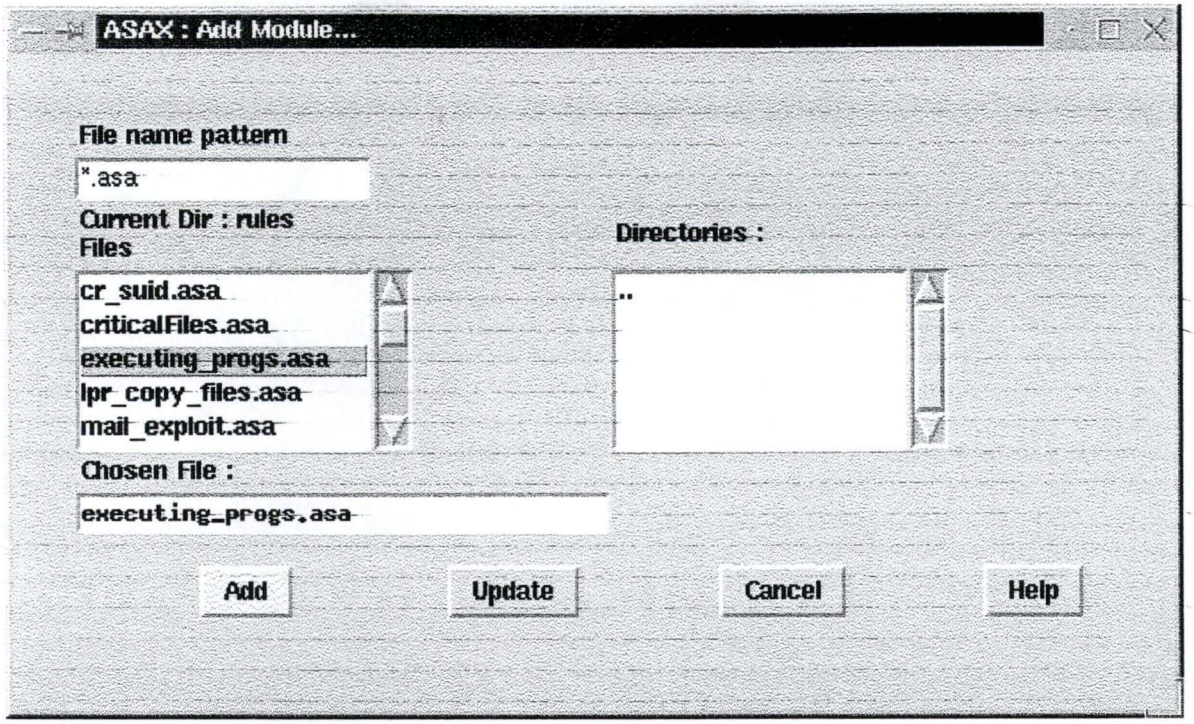


Figure 3.14 Boîte de dialogue Add module...

- *Delete Module* : cet élément permet de supprimer un module sélectionné dans le scénario courant. Il est disponible uniquement lorsque la fenêtre de scénario est active. Seule la référence à un scénario est supprimée, le fichier contenant le module lui-même n'est pas supprimé. Lorsque cet élément est sélectionné, une boîte de dialogue pour la confirmation apparaît pour éviter de supprimer le module par erreur. *S'il n'y a aucun module sélectionné, une boîte d'avertissement apparaît pour informer l'utilisateur qu'aucun module n'est sélectionné.*
- *Check* : cet élément permet de lancer le check d'un scénario ou d'un module. Lorsque la fenêtre de scénario est active, un check global est effectué, et lorsque c'est la fenêtre d'un module est active seule un check local est effectué. Cette élément ne peut être sélectionné pour les autres fenêtres.

3.5.2. Les fenêtres supplémentaires

- La fenêtre d'accueil de l'application

Cette fenêtre permet à l'utilisateur de choisir quelle action entreprendre pour utiliser ASAX. Elle comporte les boutons de commande suivant :
Le bouton ***New DDF*** *:* qui lance une fenêtre pour la création d'un fichier de description de données.
Le bouton ***Open a DDF...*** *:* qui permet de lancer une boîte de dialogue permettant de sélectionner un fichier à ouvrir.
Le bouton ***New Scenario*** : qui permet de lancer une fenêtre pour la création d'un nouveau scénario.
Le bouton ***Open a Scenario...*** : qui permet d'appeler une boîte de dialogue pour ouvrir un scénario.

Le bouton ***New Module...*** : qui permet de lancer une fenêtre pour la création d'un nouveau module.
Le bouton ***Open a Module...*** : qui permet de lancer une boîte de dialogue permettant de choisir un module existant.
Le bouton ***Abort ASAX Session*** qui permet de quitter la session ASAX si l'utilisateur ne souhaite pas utiliser ASAX.

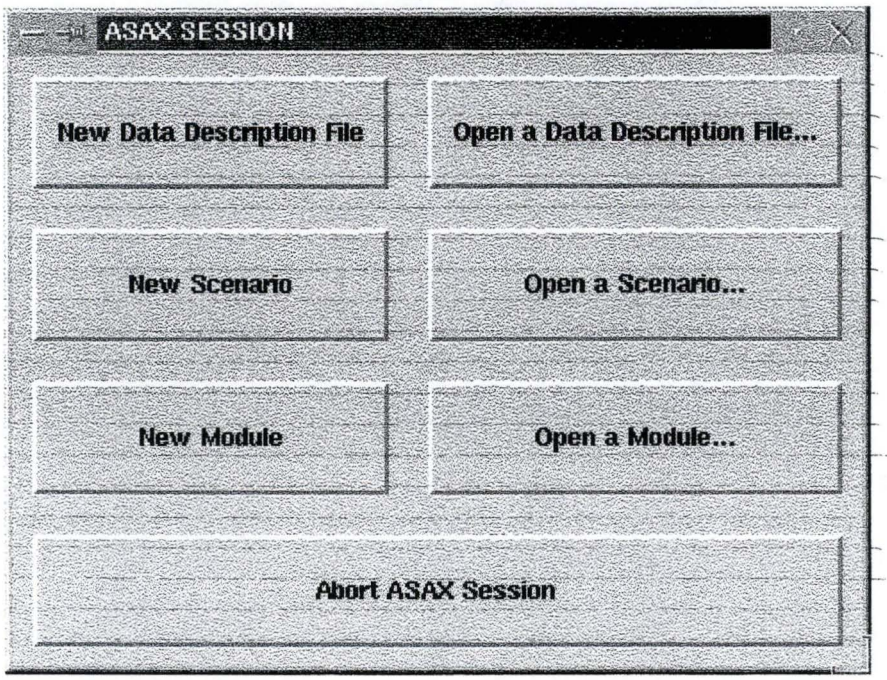


Figure 3.15 Fenêtre d'aceuil de l'application.

- Menu File :
  Dans le menu File, on pourrait ajouter les éléments suivants :
  - *Close* : cet élément du menu permet de fermer le fichier courant.
  - *Exit* : cet élément du menu permet de terminer la session ASAX.

- Menu Options :
  Dans ce menu, on pourrait ajouter les éléments suivants :
  - *C-routines library* : cet élément du menu permet d'ouvrir la fenêtre des routines C, qui affiche les routines disponibles ainsi que leur interface.
  - *Application Identifier* : Cet élément permet d'afficher une boîte de dialogue qui permet d'afficher l'identifiant de l'application. L'identifiant de l'application a au maximum 16 caractères.

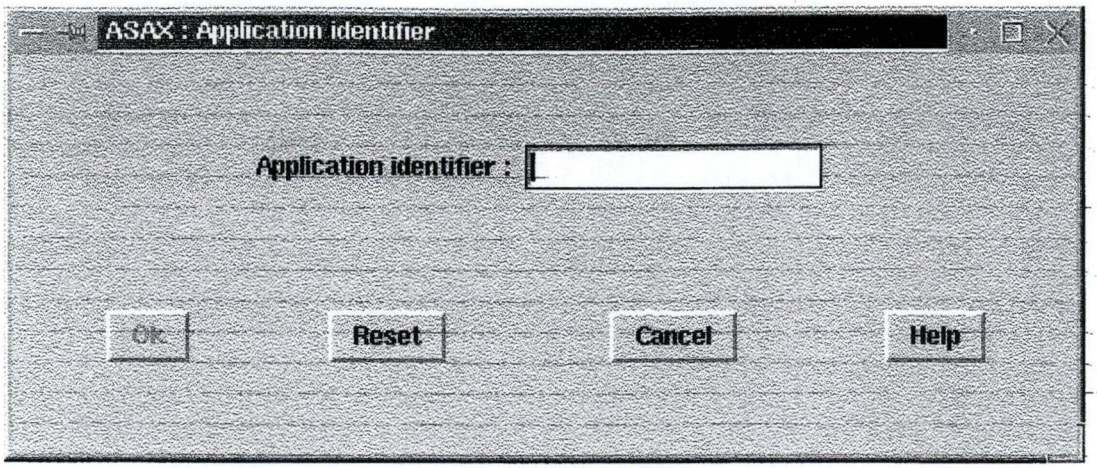


Figure 3.16 Boîte de dialogue Application identifier.

- *Editor...* : cet élément permet d'afficher une boîte de dialogue qui permet de spécifier un autre éditeur que celui fourni par l'application à utiliser pour afficher un DDF ou un module. Nous ne nous servirons pas de cet élément pour le moment.

- Le menu « Help » :
  Ce menu contient les éléments suivants :
  - *Help on context* : qui donne les informations d'aide de l'élément actif.
  - *Help on window* : qui affiche les informations d'aide pour la fenêtre active.
  - *Help on help* : permet d'afficher le texte d'aide sur les mécanismes d'aide utilisés par l'application.
  - *Help on version* : donne des informations générales sur l'application telles que le nom de l'application, la version et le copyright.

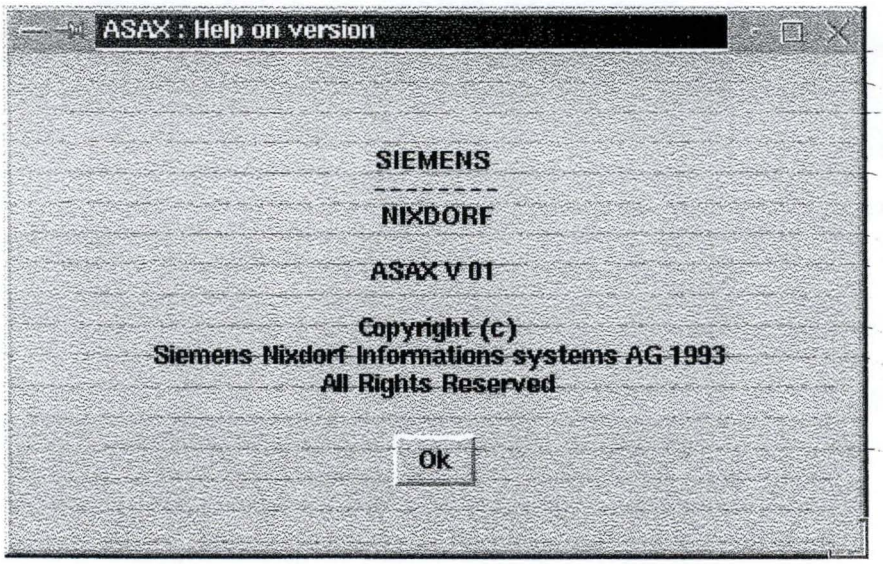


Figure 3.17 Boîte de dialogue Help on version.

- Les boîtes de message
  Une boîte de message apparaît lorsqu'ASAX doit donner à l'utilisateur certaines informations. Il y a quatre sortes d'informations : les notes, l'avertissement, l'erreur et l'échec.

- *La boite de dialogue pour les notes* : cette boîte donner les informations à l'utilisateur sans qu'il interrompe aucun traitement en cours. Il faut répondre à la boîte de dialogue pour l'enlever de l'écran.

- *La boîte de dialogue d'erreur* : cette boîte de dialogue s'affiche lorsqu'une erreur survient. Cette boîte a deux boutons de commande : le bouton Ok et le bouton Help. Le bouton Ok répond à la boîte, il la ferme et interrompt le traitement en cours. Le bouton Help donne les informations sur le type d'erreurs possibles.

- *La boîte de dialogue signalant un échec* : lorsqu'une erreur importante interrompt l'exécution normale d'ASAX, une boîte de dialogue signalant l'échec s'affiche pour informer l'utilisateur que la session a être arrêtée. Cette boîte de dialogue contient deux boutons de commande : le bouton Ok et le bouton Help. Le bouton Ok répond à la boîte de dialogue et le bouton Help donne les informations concernant le type d'erreurs possibles.

- *La boîte de dialogue d'avertissement* : cette boîte de dialogue affiche les informations d'avertissement. Il alerte l'utilisateur concernant un danger imminent pour qu'il prenne ses précautions et effectue des actions appropriées. L'avertissement consiste en une question concernant l'action qui sera faite ou pas par ASAX. Cette boîte de dialogue contient les boutons de commande suivants : le *bouton Yes*, le *bouton No*, le *bouton Cancel* et le *bouton Help*. Le *bouton Yes* est nommé en fonction de l'action qui sera entreprise par exemple *Overwrite*, pour lever l'ambiguïté sur l'action à effectuer. Le *bouton No* est nommé *No* suivi de l'action qui devait s'effectuer, ici aussi pour lever l'ambiguïté.

3.6. Evaluation de l'interface réalisé et de la méthode utilisée

Le but de l'interface est de permettre à l'utilisateur de réaliser efficacement sa tâche dans un environnement convivial. La principale qualité d'une interface est d'être utile est utilisable. Il existe des critères qui peuvent être utilisés pour évaluer la qualité d'une interface (Réf. 2). Une bonne évaluation de ces critères suppose qu'on prenne en compte le contexte dans lequel se déroule la tâche considérée. Notre application est de type commercial pour lequel on doit donc assurer un temps d'apprentissage limité, une grande rapidité d'exécution et une couverture adéquate. Nous allons citer les critères de qualité d'une interface tout en les commentant par rapport à l'interface que nous avons réalisé. Ces commentaires sont basés sur des considérations théoriques et il est souhaitable de les valider d'une façon expérimentale. Ce sont :

- Le **temps d'apprentissage** des dispositifs nécessaires à l'exécution de la tâche : Si on considère un utilisateur ayant une bonne connaissance des systèmes informatiques comme c'est le cas ici, on peut supposer que temps d'apprentissage est faible pour un utilisateur normale.
- La **rapidité d'exécution** d'une tâche : C'est un critère important pour l'efficacité d'utilisation du système. Notre interface respecte ce critère car il offre la possibilité à l'utilisateur ordinaire de lancer directement l'analyse en utilisant les programmes fournis avec le système.
- Le **taux d'erreurs** effectuées par l'utilisateur : on considère deux paramètres :
  a) Le fréquence des erreurs en distinguant deux types d'erreurs :
     ° les erreurs d'exécution liées à des défauts de manipulation

- les erreurs d'intention lorsque l'utilisateur sélectionne une commande incorrecte.

b) Le temps de correction

La gestion des erreurs de l'utilisateur n'est pas encore parfaitement maîtrisée mais elle a été prise en compte lors de la réalisation de notre interface. Par exemple, on tient à demander l'utilisateur s'il veut réellement fermer un fichier qui a été modifié sans enregistrer les modifications, les boutons de commande sont nommés en fonction de l'action qui sera effectuée pour lever l'ambiguïté, ...

- La **période de rémanence** durant laquelle un utilisateur conserve la connaissance acquise (heures, jours, ...) : ce critère est fonction directe de l'effort cognitif requis dans l'exécution de la tâche et il est fonction indirecte du temps d'apprentissage et de la fréquence d'utilisation. L'interface réalisé est très simple à utiliser et le temps d'apprentissage est faible, nous pouvons donc considérer que l'utilisateur garde pour une bonne période les connaissance acquises.
- La **satisfaction subjective à utiliser le système** : ce critère peut se traduire par un sentiment de réconfort, d'enrichissement. Ceci est difficile à évaluer mais comme notre interface est facile à utiliser nous pouvons considérer que cela contribue à satisfaire à ce critère.
- Le **degré de couverture** par rapport aux actions qui font parti de la tâche de l'utilisateur : toutes les actions nécessaires à l'analyse sont couvertes.

Nous pouvons donc conclure que notre interface possèdes les qualités suffisantes pour permettre à l'utilisateur de réaliser efficacement la tâche attendue.

La méthode que nous avons appliquée n'a pas apporté des modifications importantes de conception par rapport à l'interface décrit dans le manuel *ASAX User's Guide*. Une modification apparaît le menu *Start analysis* : nous proposons un menu en cascade pour permettre à l'utilisateur de choisir explicitement le mode d'analyse en fonction du type de format du fichier à analyser. Dans la fenêtre la boîte de dialogue liée au menu *start analysis*, l'utilisateur devra chaque fois fournir le nom du fichier à analyser et le scénario à utiliser pour l'analyse. Le *champ entry* dans lequel il faut entrer un option à passer à l'adaptateur de format lors de l'analyse en mode conversion est supprimé, car nous avons programmé de telle façon que l'on puisse appeler la commande appropriée en fonction du mode d'analyse. Cette présentation est plus conforme à l'analyse de la tâche. Les autres changements qui peuvent apparaître sont dus aux notions difficiles à programmer que nous avons présenté autrement ; nous reviendront sur ce point ultérieurement.

La méthode que nous avons appliquée constitue une base pour le concepteur des IHM, d'autres critères doivent être prise en compte notamment les critères ergonomiques, pour aboutir à des interfaces utiles et utilisables. L'avantage de cette méthode est de permettre de mener une conception basée sur une vue détaillée des différentes fonctionnalités à remplir par l'interface et permet de tenir compte du degré de connaissance de l'utilisateur en matière d'utilisation des systèmes informatiques.

3.7. Quelques commentaires techniques sur le programme donné en annexe

Dans ce qui suit, nous allons présenter quelques fonctionnalités de l'interface graphiques qui n'étaient pas évidentes à programmer et nous décriront en même temps comment nous avons procédé pour les réaliser.

- Pour éviter les erreurs de l'utilisateur, certains boutons de commandes ne peuvent être sélectionnés que lorsqu'une certaine condition est remplie. c'est le cas par exemple du bouton **Open** de la boîte de dialogue **Open...** qui peut être sélectionné uniquement lorsque l'utilisateur a sélectionné le fichier à ouvrir. Pour cela, nous

avons utilisé un mécanisme de **trace**. Nous avons associé la commande **trace** au champ qui contient le nom du fichier à ouvrir et cette commande est associée à son tour à une procédure qui est appelée chaque fois que le champ est accédé en écriture et cette procédure exécute un script qui active le bouton Open. La structure du programme est la suivante :

```
trace variable Chosf w testvar
proc testvar {nom element opr} {
     global op
     if {$nom != ""} {
     $op.open configure -state active
          }
     }
```

- Parfois, l'utilisateur est obligé de répondre à une boîte de dialogue pour pouvoir continuer à utiliser l'application. Ce genre d'interaction est appelée modale. C'est le cas de la boîte de **New...** du menu New. Nous avons utilisé la commande **grab set** suivi du nom de la fenêtre lorsque nous activons le menu New.

```
$ddf.win.file.menu add command -label "New..." \
     -command {NewWin_create;grab set $new_win}\
     -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
```

- La boîte de sélection de fichiers (par la boîte de dialogue Open) contient deux listbox qui contiennent respectivement la liste des fichiers et celle des répertoires du répertoire courant ; nous devons en même temps afficher le répertoire courant.
  Pour cela, nous trillons d'abord ordre alphabétique tout le contenu du répertoire courant et pour chaque nous testons si c'est un répertoire ou si c'est un fichier et nous insérons à l'endroit prévu. Les noms des fichiers affichés doivent avoir la même extension que celle qui est donnée dans le champs entry associé à File name Pattern ; pour cela, nous comparons l'extension de chaque fichier avec celui qui est proposé en retirant le caractère « * » et s'il y a correspondance alors nous insérons le nom de fichier dans le lisbox qui lui est réservé.
  Pour avoir le nom du répertoire courant, nous sélectionnons d'abord le chemin complet du répertoire courant, ensuite nous le décomposons en une liste en enlevant le séparateur « / » et enfin nous sélectionnons le dernier élément de la liste. Les morceaux de code qui y correspondent sont les suivants :

```
set CurrDir [pwd]
    set NewCurrDir [split [string trim $CurrDir /] /]
set textvar "Current Dir : [lindex $NewCurrDir end]\n"

proc open_dir { dir} {
    ...
ChangeDir textvar
    cd $dir
    $op.frame2.list insert end ".."
    foreach file [lsort [glob -nocomplain *] ] {
     if { [file isdirectory $file] } {
         $op.frame2.list insert end "$file/"
     } elseif { [file extension $file] == [string trim $Fnp
*]
     ...
```

- Lorsque l'utilisateur ouvre un fichier et qu'il modifie son contenu, si ensuite il ferme le fichier sans enregistrer les modifications, nous aimerions de confirmer la fermeture. Comment alors détecter qu'un fichier a été modifié ? Pour cela lorsqu'on ouvre un fichier, on mémorise son contenu et à la fermeture on compare le contenu actuel au contenu précédent. S'il y a une différence, nous appelons une boîte de dialogue qui rappelle l'utilisateur qu'il y a eu modification du contenu. La structure du programme est la suivante :

```
button .fr.btnddfnew -text "New Data Description
File"\
-command {AccFenDdf;set EDITEUR .topddf;set contddf \
[$EDITEUR.text get 1.0 end]; ...

$ddf.win.file.menu add command -label "Close" \
-command {set EDITEUR $ddf;\
VerifierChgmt $Chosfddf;set Fnp *.ddf}...

proc VerifierChgmt {fichier} {
global contddf contmdl contsce EDITEUR Chosfsce
switch $fichier {
""    {switch $EDITEUR {
          .topddf     { if {[string compare $contddf\
               [$EDITEUR.text get 1.0 end]] != 0 } {
               avertirSave
                    } else {
                    destroy $EDITEUR
                         }
                    }
...
```

avertirSave est une procédure qui implémente une boîte de dialogue d'avertissement.

- Lorsque l'utilisateur effectue une analyse comment faire pour récupérer les résultats de l'analyse qui normalement défilent à l'écran en mode commande. Pour cela nous créons un pipe. Le processus lié à ASAX exécute l'analyse d'un fichier et écrit le résultat dans le pipe, ensuite nous exécutons une procédure qui va lire dans le pipe et afficher le résultat dans une fenêtre. La structure du programme est la suivante.

```
button $str.start -text Start -state disabled\
-command {set rst .toprst;\
AffRst;destroy $str;AnalyseStd $SceFilenameStd $DdfFileNameStd}

proc AnalyseStd {MDL DDF} {
global str rst
set file_rst [open |"asaxn addf.bsm $MDL $DDF" r]
$rst.text delete 1.0 end
while {![eof $file_rst]} {
          $rst.text insert end [read $file_rst]
                    }
               close $file_rst
}
```

$SceFileNameStd est le nom du programme RUSSEL et **$DdfFileNameStd** est le nom du fichier audit. En réalité, ici nous n'utilisons pas encore le concept de scénario car nous n'avons pas encore effectuer le regroupement de nos modules en scénarios mais le principe reste le même.

Pour effacer un module dans un scénario nous sélectionnons d'abord le module à effacer puis nous utilisons la commande **delete**.

```
$sce.fr.options.menu add command -label "Delete Module" \
     -command {remove}...

proc remove { } {
     global sce
     if {[$sce.list curselection]!= ""} {
     DelMdl [$sce.list get [$sce.list curselection]]
     ...

proc effacer { } {
     global sce
     $sce.list delete [$sce.list curselection]
     }
```

Lorsque aucun module n'a été sélectionné nous rappelons à l'utilisateur qu'il faut d'abord faire la sélection. Si un module a été sélectionné nous demandons à l'utilisateur s'il veut réellement l'effacer.

Pour ajouter un module, nous vérifions d'abord s'il ne se trouve pas dans le scénario, s'il y est déjà, on le rappelle à l'utlisateur, sinon on ajoute le nouveau module.

```
     button $mdl.add -text Add -state disabled\
     -command {InsertMdl $Chosf; destroy $mdl}

     proc InsertMdl {MdlFileName} {
     global sce
     if {[regexp $MdlFileName [$sce.list get 0 end]] == 0}{
          $sce.list insert end $MdlFileName
               } else {Avertir $MdlFileName
     ...
```

- Il était prévu un scrollbar horizontal pour notre éditeur mais nous avons constaté qu'elle n'était pas active lorsque nous la placions. Nous avons alors décider d'utiliser un éditeur sans scrollbar horizontal comme fonctionne l'éditeur **emacs**.

3.8. Portabilité du langage Tcl/Tk

Le script Tcl/Tk ne peut être exécuté que s'il y un interpréteur. En plus le code est facile à comprendre et donc une application commerciale ne peut être distribuée si le code peut être lu par tout le monde. Actuellement, il existe un outil sur le web **Mktclapp** qui peut être utiliser pour générer du code C à partir du script Tcl/Tk. Les test que nous avons effectués sont concluant.

CONCLUSION

Dans ce travail, nous avons conçu et implémenté un interface graphique pour ASAX. Disposant au départ d'un interface proposé dans le manuel de l'utilisateur ASAX, nous avons appliqué une méthode de conception des interfaces graphiques vu au cours d'IHM. Nous avons constaté que les résultats obtenus ne diffèrent pas beaucoup de ce qui était proposé. Signalons que nous ne connaissons pas la méthode qui a été utilisée dans le manuel.

Le choix du langage d'implémentation Tcl/Tk a été guidé par la facilité de programmation offerte et par la puissance du langage. Le problème de portabilité ne se pose pas car actuellement il existe des outils sur le web qui permettent de générer du code C a partir du script Tcl/Tk.
Mise à part ce qui concerne le menu d'aide et les boîtes de messages renvoyés par ASAX que nous n'avons pas eu le temps de programmer, les autres fonctionnalités du manuel ont été réalisées.

Dans la suite de ce travail, on pourrait aussi envisager d'intégrer les aspects off-line d'ASAX tels que l'installation et la génération de l'exécutable.

REFERENCES BIBLIOGRAPHIQUES

1. *ASAX User's Guide*, 1994
2. Bodart F., *Cours Introductif à la conception des Interfaces Homme-Machine*, Notes de Cours, Facultés Universitaires Notre-Dame de la PAIX, Institut d'Informatique, 1999.
3. Equipe trident, *Tâche interactive de l'enregistrement d'un bon de commande client*, Facultés Universitaires Notre-Dame de la PAIX, Institut d'Informatique, 1999
4. Gérard F., *Définition et implémentation d'un langage déclaratif pour l'analyse d'audit trails*, Mémoire, Facultés Universitaires Notre-Dame de la PAIX, Institut d'Informatique, 1998.
5. Habra N., Le Charlier B., Mounji A., Mathieu I., *ASAX : Software Architecture and Rule-Based Language for Universal audit trail analysis.* Proceedings of ESORICS'92, European Symposium on Research in Computer Security, November 23-25 Toulouse, Springer-Verlag, 1992.
6. Mounji A., *Languages and Tools for Rule-Based Distributed Intrusion Detection*, PhD thesis, Computer Science Institute, University of NAMUR, 1997.
7. Ousterhout J. K., *Tcl and The Tk Toolkit*, ADDISON-WESLEY, 1994.
8. *http://www.sco.com/Technology/tcl/Tcl.html*
9. *http://www.unfix-online.com/tcltk*
10. *http://www.scriptics.com*

ANNEXES

ANNEXES

Les annexes contiennent les le code complet du programme qui a été réalisé

```
#!/usr/bin/wish8.0
proc FenAcceuil { } {
global APPLICAT app Name_APP ddf contddf contsce contmdl Chosf
set APPLICAT ASAX
wm title . "$APPLICAT SESSION"
wm minsize . 500 300
wm maxsize . 500 300
frame .fr
pack .fr
button .fr.btnddfnew -text "New Data Description File" \
      -command {AccFenDdf;set EDITEUR .topddf;set contddf \
      [$EDITEUR.text get 1.0 end];\
      set Chosf "";set Chosfddf $Chosf;unset Chosf;wm iconify .}
grid .fr.btnddfnew -column 0 -row 0 -pady 3m -padx 3m \
      -sticky we -ipady 0.5c
button .fr.btnddf -text "Open a Data Description File..."\
      -command {set Fnp *.ddf;set EDITEUR .topddf;Open_Acc;\
      open_dir "$env(HOME)";set DdfSceMdl AccFenDdf;\
      wm iconify  .}
grid .fr.btnddf -column 1 -row 0 -pady 3m -padx 3m \
      -sticky we -ipady 0.5c
button .fr.btnscenew -text "New Scenario" \
      -command {AccFenSce;set EDITEUR .topsce;set contsce \
      [$EDITEUR.list get 0 end];\
      set Chosf "";set Chosfsce $Chosf;unset Chosf;wm iconify .}
grid .fr.btnscenew -column 0 -row 1 -pady 3m -padx 3m \
      -sticky we -ipady 0.5c
button .fr.btnsce -text "Open a Scenario..." \
      -command {set Fnp *.sce;set EDITEUR .topsce;Open_Acc; \
      open_dir "$env(HOME)";set DdfSceMdl AccFenSce;\
      wm iconify  .}
grid .fr.btnsce -column 1 -row 1 -pady 3m -padx 3m \
      -sticky we -ipady 0.5c
button .fr.btnmdlnew -text "New Module" \
      -command {AccFenMdl;set EDITEUR .topp;set contmdl \
      [$EDITEUR.text get 1.0 end];\
      set Chosf "";set Chosfmdl $Chosf;unset Chosf;wm iconify .}
grid .fr.btnmdlnew -column 0 -row 2 -pady 3m -padx 3m \
      -sticky we -ipady 0.5c
button .fr.btnmdl -text "Open a Module..." \
      -command {set Fnp *.asa;set EDITEUR .topp;Open_Acc;\
      open_dir "$env(HOME)";set DdfSceMdl AccFenMdl;\
      wm iconify .}
grid .fr.btnmdl -column 1 -row 2 -pady 3m -padx 3m \
      -ipady 0.5c -sticky we
button .fr.btnexit -text "Abort ASAX Session" -command {exit} -width 10
grid .fr.btnexit -columnspan 2 -row 3 -pady 3m -padx 3m \
      -sticky we -ipady 0.5c
}
FenAcceuil
#////////////////////////////////////////////////////////////////
#----------------------------------------------------------------
#Fenetre principale de l'application
#----------------------------------------------------------------
#----------------------------------------------------------------
proc AccFenDdf { } {
```

```
global APPLICAT win EDITEUR ddf app Chosf Chosfddf
set ddf .topddf
toplevel $ddf
wm title $ddf "$APPLICAT SESSION for DDF"
wm minsize . 600 500
frame $ddf.win -relief raised  -bd 2
pack $ddf.win -fill both

#-------------------------------------------------------------
# Creation d'un menu
#-------------------------------------------------------------
menubutton $ddf.win.file -text File -underline 0\
      -menu $ddf.win.file.menu \
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*

menubutton $ddf.win.options -text Options -underline 0\
      -menu $ddf.win.options.menu \
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*

menubutton $ddf.win.help -text Help -underline 0\
      -menu $ddf.win.help.menu\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*

pack $ddf.win.file $ddf.win.options $ddf.win.help\
      -side left -fill x
menu $ddf.win.file.menu
menu $ddf.win.options.menu
menu $ddf.win.help.menu
menu $ddf.win.options.menu.analyse

$ddf.win.file.menu add command -label "New..." \
      -command {NewWin_create;grab set $new_win}\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$ddf.win.file.menu add command -label "Open..." \
      -command {set EDITEUR $ddf ;set Fnp *.ddf;Fen_Open ;\
      open_dir "$env(HOME)"}\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$ddf.win.file.menu add command -label "Close" \
      -command {set EDITEUR $ddf;\
      VerifierChgmt $Chosfddf;set Fnp *.ddf}\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$ddf.win.file.menu add separator
$ddf.win.file.menu add command -label "Save" \
      -command "Save"\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$ddf.win.file.menu add command -label "Save as..." \
      -command {set EDITEUR $ddf ;set Fnp "*.ddf";Fen_Save ;\
      print_dir "$env(HOME)"}\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$ddf.win.file.menu add separator
$ddf.win.file.menu add command -label "Exit" \
      -command {exit} \
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$ddf.win.options.menu add cascade -label "Start Analysis" \
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*\
      -menu $ddf.win.options.menu.analyse
$ddf.win.options.menu.analyse add command -label "Standard-mode
analysis..."\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*\
```

```
      -command {Fen_Start }
$ddf.win.options.menu.analyse add command -label "Conversion-mode
analysis..."\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*\
      -command {Fen_Conv }
$ddf.win.options.menu.analyse add command -label "Mixed-mode analysis..."\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*\
      -command {Fen_Mixed }
$ddf.win.options.menu add separator
$ddf.win.options.menu add command -label "Add Module..." \
      -command {Fen_Add ; mdl_dir "$env(HOME)"}\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-* -state disabled
$ddf.win.options.menu add command -label "Delete Module" \
      -command {Delete Module} -state disabled\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$ddf.win.options.menu add command -label "Check" \
      -command {Check} -state disabled\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$ddf.win.options.menu add separator
$ddf.win.options.menu add command -label "C-routines library" \
      -command {CroutinesLibrary}\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$ddf.win.options.menu add command -label "Application Identfier..." \
      -command {Fen_AppId}\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$ddf.win.options.menu add command -label "Editor..." \
      -command { Fen_Edit }\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$ddf.win.help.menu add command -label "Help on context"\
      -command {HelpOnContext} -state disabled \
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$ddf.win.help.menu add command -label "Help on window"\
      -command {HelpOnWindow} -state disabled \
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$ddf.win.help.menu add command -label "Help on help"\
      -command {HelpOnContext} -state disabled \
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$ddf.win.help.menu add command -label "Help on version"\
      -command { Fen_Helpv }\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*

#Creation d'un editeur de text

text $ddf.text -relief flat \
      -yscrollcommand "$ddf.scrv set" -bg white\
      -font *-*-medium-*-*-*-14-*-*-*-*-*-*-*

focus $ddf.text

# creation d'un scrollbar

scrollbar $ddf.scrv  -command "$ddf.text yview"
pack $ddf.text -expand 1 -fill both -side left
pack $ddf.scrv -side right -fill y
}

#////////////////////////////////////////////////////////////////
#----------------------------------------------------------------
#Fenetre principale de l'application
```

```
#-------------------------------------------------------------
#-------------------------------------------------------------
proc AccFenSce { } {
global APPLICAT win .list sce EDITEUR fen_mdl
set sce .topsce
set APPLICAT ASAX
toplevel $sce
wm title $sce "$APPLICAT SESSION for SCE"
#wm minsize . 600 500
frame $sce.fr -relief raised  -bd 2
pack $sce.fr -fill both

#-------------------------------------------------------------
# Creation d'un menu
#-------------------------------------------------------------
menubutton $sce.fr.file -text File -underline 0\
      -menu $sce.fr.file.menu \
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*

menubutton $sce.fr.options -text Options -underline 0\
      -menu $sce.fr.options.menu \
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*

menubutton $sce.fr.help -text Help -underline 0\
      -menu $sce.fr.help.menu\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*

pack $sce.fr.file $sce.fr.options $sce.fr.help\
      -side left -fill x
menu $sce.fr.file.menu
menu $sce.fr.options.menu
menu $sce.fr.help.menu
menu $sce.fr.options.menu.analyse

$sce.fr.file.menu add command -label "New..." \
      -command {NewWin_create;grab set $new_win}\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$sce.fr.file.menu add command -label "Open..." \
      -command {set EDITEUR $sce ;set Fnp "*.sce";Fen_Open ;\
      open_dir "$env(HOME)"}\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$sce.fr.file.menu add command -label "Close" \
      -command {set EDITEUR $sce;\
      VerifierChgmt $Chosfsce;set Fnp *.ddf}\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$sce.fr.file.menu add separator
$sce.fr.file.menu add command -label "Save" \
      -command "Save"\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$sce.fr.file.menu add command -label "Save as..." \
      -command {set EDITEUR $sce ; set Fnp "*.sce"; Fen_Save ;\
       print_dir "$env(HOME)"}\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$sce.fr.file.menu add separator
$sce.fr.file.menu add command -label "Exit" \
      -command {exit} \
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$sce.fr.options.menu add cascade -label "Start Analysis" \
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*\
```

```
      -menu $sce.fr.options.menu.analyse -state disabled
$sce.fr.options.menu.analyse add command -label "Standard-mode
analysis..."\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*\
      -command {Fen_Start }
$sce.fr.options.menu.analyse add command -label "Conversion-mode
analysis..."\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*\
      -command {Fen_Conv }
$sce.fr.options.menu.analyse add command -label "Mixed-mode analysis..."\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*\
      -command {Fen_Mixed }
$sce.fr.options.menu add separator
$sce.fr.options.menu add command -label "Add Module..." \
      -command {Fen_Add ; mdl_dir "$env(HOME)"}\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$sce.fr.options.menu add command -label "Delete Module" \
      -command {remove}\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$sce.fr.options.menu add command -label "Check" \
      -command {Check}\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$sce.fr.options.menu add separator
$sce.fr.options.menu add command -label "C-routines library" \
      -command {CroutinesLibrary}\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$sce.fr.options.menu add command -label "Application Identfier..." \
      -command {Fen_AppId} -state disabled\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$sce.fr.options.menu add command -label "Editor..." \
      -command { Fen_Edit }\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$sce.fr.help.menu add command -label "Help on context"\
      -command {HelpOnContext} -state disabled \
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$sce.fr.help.menu add command -label "Help on window"\
      -command {HelpOnWindow} -state disabled \
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$sce.fr.help.menu add command -label "Help on help"\
      -command {HelpOnContext} -state disabled \
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$sce.fr.help.menu add command -label "Help on version"\
      -command { Fen_Helpv }\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*

#Creation d'un editeur de text

listbox $sce.list -relief flat \
      -yscrollcommand "$sce.scrv set" -bg white\
      -font *-*-medium-*-*-*-14-*-*-*-*-*-*-*\
      -width 70 -height 10

focus $sce.list

# creation d'un scrollbar

scrollbar $sce.scrv  -command "$sce.list yview"
pack $sce.list -expand 1 -fill both -side left
pack $sce.scrv -side right -fill y
```

```
#-------------------------------------------
#Un double_click sur le nom d'un module entraîne
#l'ouverture du fichier contenant ce module
#-------------------------------------------
set fen_mdl ""
bind $sce.list <Double-ButtonPress-1> {
    set num [$sce.list get [$sce.list curselection]]
    if {$fen_mdl != ""} {
      destroy $fen_mdl
      }
    if {$num != "" } {
      AccFenMdl
      open_mdl $num
            }
break
      }
}

proc open_mdl {module} {
      global fen_mdl APPLICAT
      wm title $fen_mdl "$APPLICAT : $module"
      $fen_mdl.text delete 1.0 end
            set res [open $module]
            while {![eof $res]} {
            $fen_mdl.text insert end [read $res]
                        }
                  close $res
                  }

#-------------------------------------------------------------
#Cette gère l'effacement d'un module
#-------------------------------------------------------------
proc remove { } {
      global sce
      if {[$sce.list curselection]!= ""} {
      DelMdl [$sce.list get [$sce.list curselection]]
      } else {
      MessErr
      }
     }
#-------------------------------------------------------------
#Cette procedure permet de confirmer l'effacement d'un module
#-------------------------------------------------------------
proc DelMdl {Module} {
      global Chosf APPLICAT EDITEUR
      toplevel .delmdl
      wm title .delmdl "$APPLICAT : Message..."
      wm geometry .delmdl +150+100
      message .delmdl.mesg -width 8c -justify left -relief flat -bd 2\
      -font -Adobe-Helvetica-Medium-R-Normal--*-180-*\
      -text "\n\nDo you really want delete $Module ?"\
      -width 15c -justify center
      pack  .delmdl.mesg
      button .delmdl.btnOk -text Ok \
            -command {effacer; destroy .delmdl}
      button .delmdl.btnCancel -text Cancel \
            -command {destroy .delmdl}
      pack .delmdl.btnOk .delmdl.btnCancel -side left -padx 2c -pady 1c
      }
```

```
proc effacer { } {
      global sce
      $sce.list delete [$sce.list curselection]
      }
#-----------------------------------------------------------------
#Cette procedure est appelée si on efface alors qu'on pas
#sélectionné un module à effacer
#-----------------------------------------------------------------
proc MessErr { } {
      global Chosf APPLICAT
      toplevel .err
      wm title .err "$APPLICAT : Message..."
      wm geometry .err +150+100
      message .err.mesg -width 15c -justify left -relief flat -bd 2\
      -font -Adobe-Helvetica-Medium-R-Normal--*-180-*\
      -text "\n\nNo module was selected for removing!"\
      -justify center
      pack  .err.mesg
      button .err.btnOk -text Ok -command {destroy .err}
      pack .err.btnOk -ipadx 1c -pady 1c
      }

#////////////////////////////////////////////////////////////
#-----------------------------------------------------------
#Boite de dialogue de la fenetre New
#-----------------------------------------------------------
#-----------------------------------------------------------
proc NewWin_create { } {
global new_win CreateNew APPLICAT
set new_win .newin
toplevel $new_win -bd 2
wm title $new_win "$APPLICAT : New..."
wm maxsize $new_win 500 200
label $new_win.asobj -text "ASAX object : "
radiobutton $new_win.ddf -text "data description" -variable CreateNew \
      -value radbutt_getDDF
radiobutton $new_win.scn -text "scenario" -variable CreateNew \
      -value radbutt_getSCE
radiobutton $new_win.mdl -text "module" -variable CreateNew \
      -value radbutt_getMDL
button $new_win.ok -text Ok -command {$CreateNew}
button $new_win.rst -text Reset -command reset
button $new_win.ccl -text Cancel -command {reset ; destroy $new_win}
button $new_win.hlp -text Help -command help
grid $new_win.asobj -column 1 -row 0 -padx 1c -pady 3m -sticky w
grid $new_win.ddf -column 2 -row 0 -pady 3m -sticky w
grid $new_win.scn -column 2 -row 1 -pady 3m -sticky w
grid $new_win.mdl -column 2 -row 2 -pady 3m -sticky w
grid $new_win.ok -column 0 -row 3 -padx 1c -pady 1c
grid $new_win.rst -column 1 -row 3 -padx 1c -pady 1c
grid $new_win.ccl -column 2 -row 3 -padx 1c -pady 1c
grid $new_win.hlp -column 3 -row 3 -padx 1c -pady 1c
}

#-----------------------------------------------------------
# Renvoie le radiobutton selectionne
#-----------------------------------------------------------
proc radbutt_getDDF { } {
```

```
      global win APPLICAT ddf new_win contddf Chosfddf
            AccFenDdf
            wm title $ddf "$APPLICAT SESSION for DDF"
            $ddf.text insert end "Data description file not defined !"
            set EDITEUR .topddf
            set contddf [$EDITEUR.text get 1.0 end]
            set Chosf ""
            set Chosfddf $Chosf
            unset Chosf
            destroy $new_win
    }

proc radbutt_getSCE { } {
      global win APPLICAT sce new_win contsce Chosfsce Chosf
            AccFenSce
            wm title $sce "$APPLICAT SESSION for SCE"
            set EDITEUR .topsce
            set contsce [$EDITEUR.list get 0 end]
            set Chosf ""
            set Chosfsce $Chosf
            unset Chosf
            focus $new_win
            destroy $new_win
    }

proc radbutt_getMDL { } {
      global win APPLICAT fen_mdl new_win contmdl Chosfmdl Chosf
            AccFenMdl
            wm title $fen_mdl "$APPLICAT SESSION for MDL"
            set EDITEUR .topp
            set contmdl [$EDITEUR.text get 1.0 end]
            set Chosf ""
            set Chosfmdl $Chosf
            unset Chosf
            focus $new_win
            destroy $new_win
    }
#-------------------------------------------------------------
#-------------------------------------------------------------
#Cette procedure permet d'enregistrer les modifications
#dans un fichier ouvert
#-------------------------------------------------------------
#-------------------------------------------------------------
proc VerifierChgmt {fichier} {
      global contddf contmdl contsce EDITEUR Chosfsce
switch $fichier {
      ""    {switch $EDITEUR {
                  .topsce    { if {[string compare $contsce\
                        [$EDITEUR.list get 0 end]] != 0 } {
                        avertirSave
                              } else {
                              destroy $EDITEUR
                                    }
                     }
                  .topddf    { if {[string compare $contddf\
                        [$EDITEUR.text get 1.0 end]] != 0 } {
                        avertirSave
                              } else {
                              destroy $EDITEUR
```

```
                                    }
                              }
                        .topp     { if {[string compare $contmdl\
                              [$EDITEUR.text get 1.0 end]] != 0 } {
                              avertirSave
                                    } else {
                                    destroy $EDITEUR
                                    }
                              }
                               }
            }
      default { switch $EDITEUR {
                        .topsce { if {[string compare $contsce \
                              [$EDITEUR.list get 0 end]] != 0 } {
                              avertirModif $fichier
                                    } else {
                                    destroy $EDITEUR
                                    }
                              }
                        .topddf { if {[string compare $contddf \
                              [$EDITEUR.text get 1.0 end]] != 0 } {
                              avertirModif $fichier
                                    } else {
                                    destroy $EDITEUR
                                    }
                              }
                        .topp { if {[string compare $contmdl \
                              [$EDITEUR.text get 1.0 end]] != 0 } {
                              avertirModif $fichier
                                    } else {
                                    destroy $EDITEUR
                                    }
                              }
                        }
            }
         }
      }
#-------------------------------------------------------
#Cette procedure rappelle à l'utilisateur d'enresistrer\
#les modification avant de fermer un fichier
#-------------------------------------------------------
proc avertirModif { fichier } {
      global APPLICAT EDITEUR NomFichier
      toplevel .avertir
      wm title .avertir "$APPLICAT : Message..."
      wm geometry .avertir +150+100
      grab set .avertir
      set NomFichier $fichier
      message .avertir.mesg -width 8c -justify left -relief flat -bd 2\
      -font -Adobe-Helvetica-Medium-R-Normal--*-180-*\
      -text "\n\nThe file named $fichier has been modified, save changes ?"
      pack  .avertir.mesg
      button .avertir.btnOk -text Yes -command {overwrite $NomFichier;\
            destroy .avertir;destroy $EDITEUR}
      button .avertir.btnNo -text No \
            -command {destroy .avertir;destroy $EDITEUR}
      button .avertir.btnCancel -text Cancel -command {destroy .avertir}
      pack .avertir.btnOk .avertir.btnNo .avertir.btnCancel \
            -side left -padx 1c -pady 1c
```

```
      }
#-------------------------------------------------------
#Cette procedure est la même que la précédente sauf qu'il
#s'agit d'un nouveau fichier
#-------------------------------------------------------
proc avertirSave { } {
      global Chosf APPLICAT EDITEUR op
      toplevel .avertir
      wm title .avertir "$APPLICAT : Message..."
      wm geometry .avertir +150+100
      grab set .avertir
      message .avertir.mesg -width 8c -justify left -relief flat -bd 2\
      -font -Adobe-Helvetica-Medium-R-Normal--*-180-*\
      -text "\n\nThis document has been modified, save changes ?"
      pack  .avertir.mesg
      button .avertir.btnOk -text Yes -command {Fen_Save ;\
            print_dir  "$env(HOME)";destroy .avertir}
      button .avertir.btnNo -text No \
            -command {destroy .avertir;destroy $EDITEUR}
      button .avertir.btnCancel -text Cancel -command {destroy .avertir}
      pack .avertir.btnOk .avertir.btnNo .avertir.btnCancel \
            -side left -padx 1c -pady 1c
      }

#////////////////////////////////////////////////
#-----------------------------------------------
#Procedure pour definir une nouvelle fenetre.
#-----------------------------------------------
#-----------------------------------------------
proc AccFenMdl { } {
global APPLICAT win .text fen_mdl EDITEUR Fnp Chosfmdl Chosf
set fen_mdl .topp
toplevel $fen_mdl
wm title $fen_mdl "$APPLICAT SESSION for MDL"
wm minsize . 600 500
frame $fen_mdl.frb -relief raised  -bd 2
pack $fen_mdl.frb -fill both

#-------------------------------------------------------------
# Creation d'un menu
#-------------------------------------------------------------
menubutton $fen_mdl.frb.file -text File -underline 0\
      -menu $fen_mdl.frb.file.menu \
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*

menubutton $fen_mdl.frb.options -text Options -underline 0\
      -menu $fen_mdl.frb.options.menu \
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*

menubutton $fen_mdl.frb.help -text Help -underline 0\
      -menu $fen_mdl.frb.help.menu\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*

pack $fen_mdl.frb.file $fen_mdl.frb.options $fen_mdl.frb.help\
      -side left -fill x
menu $fen_mdl.frb.file.menu
menu $fen_mdl.frb.options.menu
menu $fen_mdl.frb.help.menu
menu $fen_mdl.frb.options.menu.analyse
```

```
$fen_mdl.frb.file.menu add command -label "New..." \
      -command {NewWin_create;grab set $new_win}\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$fen_mdl.frb.file.menu add command -label "Open..." \
      -command {set EDITEUR $fen_mdl;set Fnp *.asa;Fen_Open ;\
      open_dir "$env(HOME)"}\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$fen_mdl.frb.file.menu add command -label "Close" \
      -command {set EDITEUR $fen_mdl;\
      VerifierChgmt $Chosfmdl;set Fnp *.asa}\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$fen_mdl.frb.file.menu add separator
$fen_mdl.frb.file.menu add command -label "Save" \
      -command "Save"\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$fen_mdl.frb.file.menu add command -label "Save as..." \
      -command {set EDITEUR $fen_mdl;set Fnp *.asa;Fen_Save ;\
      print_dir "$env(HOME)"}\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$fen_mdl.frb.file.menu add separator
$fen_mdl.frb.file.menu add command -label "Exit" \
      -command {exit} \
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$fen_mdl.frb.options.menu add cascade -label "Start Analysis" \
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-* -state disabled\
      -menu $fen_mdl.frb.options.menu.analyse
$fen_mdl.frb.options.menu.analyse add command -label "Standard-mode
analysis..."\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*\
      -command {Fen_Start }
$fen_mdl.frb.options.menu.analyse add command -label "Conversion-mode
analysis..."\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*\
      -command {Fen_Conv }
$fen_mdl.frb.options.menu.analyse add command -label "Mixed-mode
analysis..."\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*\
      -command {Fen_Mixed }
$fen_mdl.frb.options.menu add separator
$fen_mdl.frb.options.menu add command -label "Add Module..." \
      -command {Fen_Add ; mdl_dir "$env(HOME)"}\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-* -state disabled
$fen_mdl.frb.options.menu add command -label "Delete Module" \
      -command {Delete Module} -state disabled\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$fen_mdl.frb.options.menu add command -label "Check" \
      -command {Check} -state disabled\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$fen_mdl.frb.options.menu add separator
$fen_mdl.frb.options.menu add command -label "C-routines library" \
      -command {CroutinesLibrary}\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$fen_mdl.frb.options.menu add command -label "Application Identfier..." \
      -command {Fen_AppId} -state disabled\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$fen_mdl.frb.options.menu add command -label "Editor..." \
      -command { Fen_Edit }\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
```

```
$fen_mdl.frb.help.menu add command -label "Help on context"\
      -command {HelpOnContext} -state disabled \
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$fen_mdl.frb.help.menu add command -label "Help on window"\
      -command {HelpOnWindow} -state disabled \
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$fen_mdl.frb.help.menu add command -label "Help on help"\
      -command {HelpOnContext} -state disabled \
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$fen_mdl.frb.help.menu add command -label "Help on version"\
      -command { Fen_Helpv }\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*

#Creation d'un editeur de text

text $fen_mdl.text -relief flat \
      -yscrollcommand "$fen_mdl.scrv set" -bg white\
      -font *-*-medium-*-*-*-14-*-*-*-*-*-*-*

focus $fen_mdl.text

# creation d'un scrollbar

scrollbar $fen_mdl.scrv  -command "$fen_mdl.text yview"
pack $fen_mdl.text -expand 1 -fill both -side left
pack $fen_mdl.scrv -side right -fill y
}
#-----------------------------------------------------------------
#Annule la selection du radiobutton
#-----------------------------------------------------------------
proc reset { } {
global new_win
$new_win.ddf  deselect
$new_win.scn  deselect
$new_win.mdl  deselect
}

#-----------------------------------------------------
#Cette procedure est appelée par la fenêtre d'acceuil
#-----------------------------------------------------
proc Open_Acc { } {
global op Fnp Chosf APPLICAT
set op .open
toplevel $op -bd 1c
wm title $op "$APPLICAT : Open..."
wm minsize $op 600 380
wm maxsize $op 600 400
wm geometry $op +150+100

trace variable Chosf w testvar

label $op.fnp -text "File name pattern"
grid $op.fnp -column 0 -row 0 -sticky w
entry $op.entfnp -textvariable Fnp -background white
#set Fnp "*.asa"
grid $op.entfnp -column 0 -row 1 -sticky w
label $op.cdir -textvariable textvar
grid $op.cdir -column 0 -row 2 -sticky wn
label $op.files -text Files
```

```
grid $op.files -column 0 -row 2 -sticky ws
label $op.direct -text "Directories : "
grid $op.direct -column 2 -row 2 -sticky w
frame $op.frame1
grid $op.frame1 -column 0 -row 3 -sticky w
scrollbar $op.frame1.scr -command {$op.frame1.list yview}
grid $op.frame1.scr -column 1 -row 3 -sticky n,s
listbox $op.frame1.list -selectmode single -background white\
      -yscrollcommand {$op.frame1.scr set} -width 20 -height 5
grid $op.frame1.list -column 0 -row 3 -sticky w

frame $op.frame2
grid $op.frame2 -column 2 -row 3 -sticky w
scrollbar $op.frame2.scroll -command {$op.frame2.list yview}
grid $op.frame2.scroll -column 3 -row 3 -sticky n,s
listbox $op.frame2.list -selectmode single -background white\
      -yscrollcommand {$op.frame2.scroll set}  -width 20 -height 5
grid $op.frame2.list -column 2 -row 3 -sticky w

bind $op.frame2.list <Double-ButtonPress-1> {
    $op.entchosf delete 0 end
    set num [$op.frame2.list curselection]
    if {$num != "" } {
      open_dir [$op.frame2.list get $num]
set CurrDir [pwd]
    set NewCurrDir [split [string trim $CurrDir /] /]
set textvar "Current Dir : [lindex $NewCurrDir end]\n"
    }
break
}

bind $op.frame1.list <Double-ButtonPress-1> {
    set num [$op.frame1.list curselection]
    if {$num != "" } {
      FileName [$op.frame1.list get $num]
    }
break
}

label $op.chosf -text "Chosen File :"
grid $op.chosf -column 0 -row 4 -sticky w
entry $op.entchosf -textvariable Chosf -background white
grid $op.entchosf -columnspan 2 -row 5 -sticky w,e
button $op.open -text Open -state disabled \
      -command {$DdfSceMdl;lecture $Chosf ;\
      wm title $EDITEUR "$APPLICAT : $Chosf"; destroy $op}
grid $op.open -column 0 -row 6  -pady 5m -padx 5m
button $op.update -text Update -command {open_dir $env(HOME) ;\
      $op.entchosf delete 0 end  ;\
      $op.open configure -state disabled }
grid $op.update -column 1 -row 6 -padx 5m
button $op.cancel -text Cancel -command { $op.entchosf delete 0 20;\
      destroy $op;unset Chosf;wm deiconify  . }
grid $op.cancel -column 2 -row 6 -padx 5m
button $op.help -text Help
grid $op.help -column 3 -row 6 -padx 5m
focus $op.entfnp

#--------------------------------------------------------------------
```

```
#Permet d'afficher le nom du fichier selectionne
#-------------------------------------------------------------------
proc FileName { nomFichier } {
global op , Chosf
set Chosf $nomFichier
bind $op.entchosf $Chosf
}
#-------------------------------------------------------------------
#Permet de tester la modification en ecriture du nom du fichier
#cette procedure est appele par la commande trace
#-------------------------------------------------------------------
proc testvar {nom element opr} {
global op
    if {$nom != ""} {
 $op.open configure -state active
   }
}
#-------------------------------------------------------------------
#Permet d'afficher le repertoire courant lors browsing
#-------------------------------------------------------------------
proc ChangeDir {dir} {
global op
bind $op.cdir $dir
}

bind $op.entfnp <Return> { bind $op.entfnp $Fnp ; open_dir "$env(HOME)";\
                           $op.entfnp configure -state disabled ; \
                      $op.entchosf delete 0 end }

bind $op.entfnp <ButtonPress>  {$op.entfnp configure -state normal}
}
#-------------------------------------------------------------------
#Affichage de repertoires et de fichiers d'un repertoire selectionnes
#-------------------------------------------------------------------
proc open_dir { dir} {
    global op , Fnp , file , env , textvar
    $op.frame1.list delete 0 end
    $op.frame2.list delete 0 end
    bind $op.entfnp $Fnp
set CurrDir $dir
    set DIR [split [string trim $CurrDir /] /]
    set textvar "Current Dir : [lindex $DIR end]\n"
ChangeDir textvar
    cd $dir
    $op.frame2.list insert end ".."
    foreach file [lsort [glob -nocomplain *] ] {
      if { [file isdirectory $file] } {
          $op.frame2.list insert end "$file/"
      } elseif { [file extension $file] == [string trim $Fnp *] } {
        $op.frame1.list insert end "$file"
      }
    }
}

#////////////////////////////////////////////////////////////
#-----------------------------------------------------------
#Boite de dialogue de la fenetre open
#-----------------------------------------------------------
#-----------------------------------------------------------
```

```
proc Fen_Open { } {
global op Fnp Chosf APPLICAT
set op .open
toplevel $op -bd 1c
wm title $op "$APPLICAT : Open..."
wm minsize $op 600 380
wm maxsize $op 600 400
wm geometry $op +150+100

trace variable Chosf w testvar

label $op.fnp -text "File name pattern"
grid $op.fnp -column 0 -row 0 -sticky w
entry $op.entfnp -textvariable Fnp -background white
#set Fnp "*.asa"
grid $op.entfnp -column 0 -row 1 -sticky w
label $op.cdir -textvariable textvar
grid $op.cdir -column 0 -row 2 -sticky wn
label $op.files -text Files
grid $op.files -column 0 -row 2 -sticky ws
label $op.direct -text "Directories : "
grid $op.direct -column 2 -row 2 -sticky w
frame $op.frame1
grid $op.frame1 -column 0 -row 3 -sticky w
scrollbar $op.frame1.scr -command {$op.frame1.list yview}
grid $op.frame1.scr -column 1 -row 3 -sticky n,s
listbox $op.frame1.list -selectmode single -background white\
      -yscrollcommand {$op.frame1.scr set} -width 20 -height 5
grid $op.frame1.list -column 0 -row 3 -sticky w

frame $op.frame2
grid $op.frame2 -column 2 -row 3 -sticky w
scrollbar $op.frame2.scroll -command {$op.frame2.list yview}
grid $op.frame2.scroll -column 3 -row 3 -sticky n,s
listbox $op.frame2.list -selectmode single -background white\
      -yscrollcommand {$op.frame2.scroll set}  -width 20 -height 5
grid $op.frame2.list -column 2 -row 3 -sticky w

bind $op.frame2.list <Double-ButtonPress-1> {
    $op.entchosf delete 0 end
    set num [$op.frame2.list curselection]
    if {$num != "" } {
      open_dir [$op.frame2.list get $num]
set CurrDir [pwd]
    set NewCurrDir [split [string trim $CurrDir /] /]
set textvar "Current Dir : [lindex $NewCurrDir end]\n"
    }
#break
}

bind $op.frame1.list <Double-ButtonPress-1> {
    set num [$op.frame1.list curselection]
    if {$num != "" } {
      FileName [$op.frame1.list get $num]
    }
break
}

label $op.chosf -text "Chosen File :"
```

```
grid $op.chosf -column 0 -row 4 -sticky w
entry $op.entchosf -textvariable Chosf -background white
grid $op.entchosf -columnspan 2 -row 5 -sticky w,e
button $op.open -text Open -state disabled \
      -command {lecture $Chosf ; \
      wm title $EDITEUR "$APPLICAT : $Chosf";destroy $op}
grid $op.open -column 0 -row 6  -pady 5m -padx 5m
button $op.update -text Update -command {open_dir $env(HOME) ;\
      $op.entchosf delete 0 end  ;\
      $op.open configure -state disabled }
grid $op.update -column 1 -row 6 -padx 5m
button $op.cancel -text Cancel -command { $op.entchosf delete 0 20 ; \
      destroy $op }
grid $op.cancel -column 2 -row 6 -padx 5m
button $op.help -text Help
grid $op.help -column 3 -row 6 -padx 5m
focus $op.entfnp

#---------------------------------------------------------------------
#Permet d'afficher le nom du fichier selectionne
#---------------------------------------------------------------------
proc FileName { nomFichier } {
global op , Chosf
set Chosf $nomFichier
bind $op.entchosf $Chosf
}
#---------------------------------------------------------------------
#Permet de tester la modification en ecriture du nom du fichier
#cette procedure est appele par la commande trace
#---------------------------------------------------------------------
proc testvar {nom element opr} {
global op
    if {$nom != ""} {
 $op.open configure -state active
   }
}
#---------------------------------------------------------------------
#Permet d'afficher le repertoire courant lors browsing
#---------------------------------------------------------------------
proc ChangeDir {dir} {
global op
bind $op.cdir $dir
}

bind $op.entfnp <Return> { bind $op.entfnp $Fnp ; open_dir "$env(HOME)";\
                           $op.entfnp configure -state disabled ; \
                      $op.entchosf delete 0 end }

bind $op.entfnp <ButtonPress>  {$op.entfnp configure -state normal}
}
#---------------------------------------------------------------------
#Affichage de repertoires et de fichiers d'un repertoire selectionnes
#---------------------------------------------------------------------
proc open_dir { dir} {
    global op , Fnp , file , env , textvar
    $op.frame1.list delete 0 end
    $op.frame2.list delete 0 end
    bind $op.entfnp $Fnp
set CurrDir $dir
```

```
    set DIR [split [string trim $CurrDir /] /]
    set textvar "Current Dir : [lindex $DIR end]\n"
ChangeDir textvar
    cd $dir
    $op.frame2.list insert end ".."
    foreach file [lsort [glob -nocomplain *] ] {
      if { [file isdirectory $file] } {
          $op.frame2.list insert end "$file/"
      } elseif { [file extension $file] == [string trim $Fnp *] } {
        $op.frame1.list insert end "$file"
      }
    }
}
#////////////////////////////////////////////////
#------------------------------------------------
#Cette procedure lit les données d'un fichier
#------------------------------------------------
proc lecture {fichier} {
      global op EDITEUR contddf contsce Chosf Chosfddf\
            Chosfmdl Chosfsce contmdl
switch $EDITEUR {
      .topddf {$EDITEUR.text delete 1.0 end
            set res [open $fichier]
            while {![eof $res]} {
            $EDITEUR.text insert end [read $res]
                        }
                  close $res
            set contddf [$EDITEUR.text get 1.0 end]
            set Chosfddf $Chosf
            unset Chosf
                  }
      .topp {$EDITEUR.text delete 1.0 end
            set res [open $fichier]
            while {![eof $res]} {
            $EDITEUR.text insert end [read $res]
                        }
                  close $res
            set contmdl [$EDITEUR.text get 1.0 end]
            set Chosfmdl $Chosf
            unset Chosf
                  }
      .topsce {$EDITEUR.list delete 0 end
            set res [open $fichier]
            while {![eof $res]} {
            foreach i [read $res] {
                  $EDITEUR.list insert end $i
                        }
                  }
            close $res
            set contsce [$EDITEUR.list get 0 end]
            set Chosfsce $Chosf
            unset Chosf
                  }
      }
}
#////////////////////////////////////////////////////////////////////////
#------------------------------------------------------------------------
#Boite de dialogue du menu Save as
#C'est le meme code que pour le menu open a quelques changement pres
```

```
#-------------------------------------------------------------------
#-------------------------------------------------------------------
proc Fen_Save { } {
global sav Fnp Chosf APPLICAT EDITEUR
set sav .save
catch {destroy $sav}
toplevel $sav -bd 1c
wm title $sav "$APPLICAT : Save as..."
wm minsize $sav 600 380
wm maxsize $sav 600 400
wm geometry $sav +150+100

trace variable Chosf w testvar

label $sav.fnp -text "File name pattern"
grid $sav.fnp -column 0 -row 0 -sticky w
entry $sav.entfnp -textvariable Fnp -background white
#set Fnp "*.asa"
grid $sav.entfnp -column 0 -row 1 -sticky w
label $sav.cdir -textvariable textvar
grid $sav.cdir -column 0 -row 2 -sticky wn
label $sav.files -text Files
grid $sav.files -column 0 -row 2 -sticky ws
label $sav.direct -text "Directories : "
grid $sav.direct -column 2 -row 2 -sticky w
frame $sav.frame1
grid $sav.frame1 -column 0 -row 3 -sticky w
scrollbar $sav.frame1.scr -command {$sav.frame1.list yview}
grid $sav.frame1.scr -column 1 -row 3 -sticky n,s
listbox $sav.frame1.list -selectmode single -background white\
      -yscrollcommand {$sav.frame1.scr set} -width 20 -height 5
grid $sav.frame1.list -column 0 -row 3 -sticky w

frame $sav.frame2
grid $sav.frame2 -column 2 -row 3 -sticky w
scrollbar $sav.frame2.scroll -command {$sav.frame2.list yview}
grid $sav.frame2.scroll -column 3 -row 3 -sticky n,s
listbox $sav.frame2.list -selectmode single -background white\
      -yscrollcommand {$sav.frame2.scroll set}  -width 20 -height 5
grid $sav.frame2.list -column 2 -row 3 -sticky w

bind $sav.frame2.list <Double-ButtonPress-1> {
    $sav.entchosf delete 0 end

    set num [$sav.frame2.list curselection]
    if {$num != "" } {
      print_dir [$sav.frame2.list get $num]
set CurrDir [pwd]
    set NewCurrDir [split [string trim $CurrDir /] /]
set textvar "Current Dir : [lindex $NewCurrDir end]\n"
    }
break
}

bind $sav.frame1.list <Double-ButtonPress-1> {
    set num [$sav.frame1.list curselection]
    if {$num != "" } {
      FileName [$sav.frame1.list get $num]
    }
```

```
break
}

label $sav.chosf -text "Chosen File :"
grid $sav.chosf -column 0 -row 4 -sticky w
entry $sav.entchosf -textvariable Chosf -background white
grid $sav.entchosf -columnspan 2 -row 5 -sticky w,e
button $sav.save -text Save -state disabled \
      -command {enregistrer $Chosf ; destroy $sav}
grid $sav.save -column 0 -row 6  -pady 5m -padx 5m
button $sav.update -text Update -command {print_dir "$env(HOME)" ;\
       $sav.entchosf delete 0 end ; \
        $sav.save configure -state disabled }
grid $sav.update -column 1 -row 6 -padx 5m
button $sav.cancel -text Cancel -command {$sav.entchosf delete 0 end ; \
      destroy $sav}
grid $sav.cancel -column 2 -row 6 -padx 5m
button $sav.help -text Help
grid $sav.help -column 3 -row 6 -padx 5m
focus $sav.entfnp

proc FileName { nomFichier } {
global sav , Chosf
set Chosf $nomFichier
bind $sav.entchosf $Chosf
}

proc testvar {nom element opr} {
global sav
    if {$nom != ""} {
 $sav.save configure -state active
   }
}
proc ChangeDir {dir} {
global sav
bind $sav.cdir $dir
}

bind $sav.entfnp <Return> { bind $sav.entfnp $Fnp ; print_dir
"$env(HOME)";\
                            $sav.entfnp configure -state disabled ;\
                        $sav.entchosf delete 0 end }

bind $sav.entfnp <ButtonPress>  {$sav.entfnp configure -state normal}
}

proc print_dir { dir} {
    global sav , Fnp , file , textvar , env
    $sav.frame1.list delete 0 end
    $sav.frame2.list delete 0 end
    bind $sav.entfnp $Fnp
set CurrDir $dir
    set DIR [split [string trim $CurrDir /] /]
    set textvar "Current Dir : [lindex $DIR end]\n"
ChangeDir textvar
    cd $dir
    $sav.frame2.list insert end ".."
    foreach file [lsort [glob -nocomplain *] ] {
      if { [file isdirectory $file] } {
```

```
            $sav.frame2.list insert end "$file/"
        } elseif { [file extension $file] == [string trim $Fnp *] } {
          $sav.frame1.list insert end "$file"
        }
    }
proc ChangeDir {dir} {
global sav
bind $sav.cdir $dir
}
}
#-----------------------------------------------------------------
#procedure de création d'un fichier et d'enregistrement de données
#-----------------------------------------------------------------
proc enregistrer {fichier} {
      global EDITEUR
      if { [file exists $fichier] == 1} {
            WriteMess $fichier
            grab set .frc
      } else {
switch $EDITEUR {
      .topddf { set file [open $fichier {w+}]
            puts -nonewline $file [$EDITEUR.text get 1.0 end]
            close $file
            }
      .topp {set file [open $fichier {w+}]
            puts -nonewline $file [$EDITEUR.text get 1.0 end]
            close $file
            }
      .topsce {set file [open $fichier {w+}]
            puts -nonewline $file [$EDITEUR.list get 0 end]
            close $file
            }
      }
    }
}
#////////////////////////////////////////////////////////////////
#Cette procedure est appelee si le fichier a enregistrer existe
#----------------------------------------------------------------
#----------------------------------------------------------------
proc WriteMess {fichier} {
      global Chosf APPLICAT EDITEUR
      toplevel .frc
      wm title .frc "$APPLICAT : Message..."
      wm geometry .frc +150+100
      message .frc.mesg -width 8c -justify left -relief flat -bd 2\
      -font -Adobe-Helvetica-Medium-R-Normal--*-180-*\
      -text "\n\nThe file named $Chosf alredy exists, overwrite it ?"
      pack  .frc.mesg
      button .frc.btnOk -text Ok -command {overwrite $Chosf; destroy .frc}
      button .frc.btnCancel -text Cancel \
            -command {destroy .frc}
      pack .frc.btnOk .frc.btnCancel -side left -padx 2c -pady 1c
      }

proc overwrite {fichier} {
      global EDITEUR
switch $EDITEUR {
      .topddf { set file [open $fichier {w+}]
            puts -nonewline $file [$EDITEUR.text get 1.0 end]
```

```
            close $file
            }
      .topp {set file [open $fichier {w+}]
            puts -nonewline $file [$EDITEUR.text get 1.0 end]
            close $file
            }
      .topsce {set file [open $fichier {w+}]
            puts -nonewline $file [$EDITEUR.list get 0 end]
            close $file
            }
      }
}

#////////////////////////////////////////////////////////////////////
#--------------------------------------------------------------------
#Boite de dialogue du menu start analysis
#Cette fenetre est relative a l'analyse en stanfard mode
#--------------------------------------------------------------------
#--------------------------------------------------------------------
proc Fen_Start { } {
global str
set str .standard
catch {destroy $str}
toplevel $str -bd 4
wm title $str "Start analysis"
wm iconname $str "filebox"

trace variable SceFileNameStd w testvar

label $str.msg -wraplength 4i -justify left \
      -text "Standard mode ASAX application."
grid $str.msg -column 1 -row 0 -sticky we

button $str.start -text Start -state disabled\
      -command {set rst .toprst;\
      AffRst;destroy $str;AnalyseStd $DdfFileNameStd}
grid $str.start -column 0 -row 3 -pady 1c -sticky e
button $str.update -text "Update" \
      -command {$str.ent delete 0 end;$str.entsce delete 0 end;\
      $str.start configure -state active}
grid $str.update -column 1 -row 3
button $str.cancel -text "Cancel" \
      -command "destroy $str"
grid $str.cancel -column 2 -row 3 -sticky w
button $str.help -text "Help" -command ""
grid $str.help -column 3 -row 3 -sticky w -padx 1c

    label $str.lab -text "Select a ddf file : " -anchor e
grid $str.lab -column 0 -row 1 -sticky we -pady 0.5c
    entry $str.ent -textvariable DdfFileNameStd -width 20 \
      -background white
grid $str.ent -column 1 -row 1 -sticky we
    button $str.browse -text "Browse ..." \
      -command "fileDialogDdfStd $str $str.ent ddf"
grid $str.browse -column 2 -row 1 -sticky w

    label $str.labsce -text "Select a sce file : " -anchor e
grid $str.labsce -column 0 -row 2 -sticky we -pady 0.5c
    entry $str.entsce -width 20 -textvariable SceFileNameStd\
```

```
      -background white
grid $str.entsce -column 1 -row 2 -sticky we
    button $str.browsesce -text "Browse ..." \
      -command "fileDialogSceStd $str $str.entsce sce"
grid $str.browsesce -column 2 -row 2 -sticky w

proc testvar {nom element opr} {
global str
    if {$nom != ""} {
    $str.start configure -state active
   }
}

proc fileDialogDdfStd {w ent operation} {
    #   Type names            Extension(s)      Mac File Type(s)
    #
    #---------------------------------------------------------
    set types {
      {"ddf files"            {.ddf}      }
      {"All files"            *}
    }
    if {$operation == "ddf"} {
      set file [tk_getOpenFile -filetypes $types -parent $w]
    } else {
      set file [tk_getOpenFile -filetypes $types -parent $w]
    }
    if [string compare $file ""] {
      $ent delete 0 end
      $ent insert 0 $file
      $ent xview end
    }
}


proc fileDialogSceStd {w ent operation} {
    #   Type names            Extension(s)      Mac File Type(s)
    #
    #---------------------------------------------------------
    set types {
      {"Scenrio files"  {.sce}      }
      {"Module files"         {.asa}      }
      {"All files"            *}
    }
    if {$operation == "ddf"} {
      set file [tk_getOpenFile -filetypes $types -parent $w]
    } else {
      set file [tk_getOpenFile -filetypes $types -parent $w]
    }
    if [string compare $file ""] {
      $ent delete 0 end
      $ent insert 0 $file
      $ent xview end
    }
  }
 }

#-----------------------------------------------
#-----------------------------------------------
#Cette procedure lance la procedure d'analyse
```

```
#et renvoie le résultat
#-----------------------------------------------
#-----------------------------------------------
proc AnalyseStd {command } {
global str rst
set file_rst [open |$command r]
$rst.text delete 1.0 end
while {![eof $file_rst]} {
            $rst.text insert end [read $file_rst]
                        }
                  close $file_rst
}

#////////////////////////////////////////////////////////////////////////
#------------------------------------------------------------------------
#Boite de dialogue du menu start analysis
#Cette fenetre est relative a l'analyse en conversion-mode
#------------------------------------------------------------------------
#------------------------------------------------------------------------
proc Fen_Conv { } {
global f3 f4 f5 conv
set conv .conversion
catch {destroy $conv}
toplevel $conv -bd 4
wm title $conv "Start analysis"
wm iconname $conv "filebox"

label $conv.labconv -wraplength 4i -justify left \
      -text "Conversion-mode ASAX application."
grid  $conv.labconv -column 1 -row 0 -sticky we

button $conv.startconv -text Start \
      -command {set rst .toprst;\
      AffRst;destroy $conv;AnalyseStd $DdfFileNameconv }
grid  $conv.startconv -column 0 -row 3 -sticky e
button $conv.updateconv -text "Update" \
      -command {$conv.entconvsce delete 0 end;\
      $conv.entconv delete 0 end;\
      $conv.entconvopt delete 0 end}
grid  $conv.updateconv -column 1 -row 3 -pady 0.5c
button $conv.cancelconv -text "Cancel" \
      -command "destroy $conv"
grid  $conv.cancelconv -column 2 -row 3 -pady 0.5c -sticky w
button $conv.helpconv -text "Help" -command ""
grid  $conv.helpconv -column 3 -row 3 -padx 1c -sticky w

    label $conv.labconvddf -text "Select a ddf file : " -anchor e
grid  $conv.labconvddf -column 0 -row 1 -sticky w -pady 0.5c
    entry $conv.entconv -width 20 -textvariable DdfFileNameconv \
      -background white
grid  $conv.entconv -column 1 -row 1 -sticky we -pady 0.5c
    button $conv.browseconv -text "Browse ..." \
      -command "fileDialogDdfconv $conv $conv.entconv ddf"
grid  $conv.browseconv -column 2 -row 1 -pady 0.5c

    label $conv.labconvsce -text "Select a sce file : " -anchor e
grid  $conv.labconvsce -column 0 -row 2 -sticky w -pady 0.5c
    entry $conv.entconvsce -width 20 -textvariable SceFileNameconv\
      -background white
```

```
grid  $conv.entconvsce -column 1 -row 2 -sticky we -pady 0.5c
    button $conv.browseconvsce -text "Browse ..." \
      -command "fileDialogSceconv $conv $conv.entconvsce sce"
grid  $conv.browseconvsce -column 2 -row 2 -pady 0.5c

proc fileDialogDdfconv {w ent operation} {
    #   Type names           Extension(s)      Mac File Type(s)
    #
    #---------------------------------------------------------
    set types {
      {"ddf files"           {.ddf}      }
      {"All files"           *}
    }
    if {$operation == "ddf"} {
      set file [tk_getOpenFile -filetypes $types -parent $w]
    } else {
      set file [tk_getOpenFile -filetypes $types -parent $w]
    }
    if [string compare $file ""] {
      $ent delete 0 end
      $ent insert 0 $file
      $ent xview end
    }
}

proc fileDialogSceconv {w ent operation} {
    #   Type names           Extension(s)      Mac File Type(s)
    #
    #---------------------------------------------------------
    set types {
      {"Scenrio files"  {.sce}      }
      {"Module files"        {.asa}      }
      {"All files"           *}
    }
    if {$operation == "ddf"} {
      set file [tk_getOpenFile -filetypes $types -parent $w]
    } else {
      set file [tk_getOpenFile -filetypes $types -parent $w]
    }
    if [string compare $file ""] {
      $ent delete 0 end
      $ent insert 0 $file
      $ent xview end
    }
   }
 }

#////////////////////////////////////////////////////////////////////
#--------------------------------------------------------------------
#Boite de dialogue du menu start analysis
#Cette fenetre est relative a l'analyse en mixed-mode
#--------------------------------------------------------------------
#--------------------------------------------------------------------
proc Fen_Mixed { } {
global mix
set mix .mixed
catch {destroy $mix}
toplevel $mix -bd 4
wm title $mix "Start analysis"
```

```
wm iconname $mix "filebox"

label $mix.msg -wraplength 4i -justify left \
      -text "Mixed-mode ASAX application."
grid $mix.msg -column 1 -row 0

button $mix.start -text Start \
      -command {set rst .toprst;\
      AffRst;destroy $mix;AnalyseStd $DdfFileNameMix }
grid  $mix.start -column 0 -row 4 -pady 0.5c -sticky e
button $mix.update -text "Update" \
      -command {$f6.rdba deselect;$f7.ent delete 0 end;\
      $f8.ent delete 0 end;$f9.ent delete 0 end;\
      $f6.rdbb deselect}
grid  $mix.update -column 1 -row 4 -pady 0.5c
button $mix.cancel -text "Cancel" \
      -command "destroy $mix"
grid  $mix.cancel -column 2 -row 4 -pady 0.5c -sticky w
button $mix.help -text "Help" -command ""
grid  $mix.help -column 3 -row 4 -pady 0.5c -sticky w -padx 1c

    label $mix.labmode -text "Mode : " -anchor e
grid  $mix.labmode -column 0 -row 1 -pady 0.5c -sticky e
    radiobutton $mix.rdba -text "standard" \
      -variable stdconv -value moda
grid  $mix.rdba -column 1 -row 1 -pady 0.5c -sticky we
    radiobutton $mix.rdbb -text "conversion" \
      -variable stdconv -value modb
grid  $mix.rdbb -column 2 -row 1 -pady 0.5c -sticky w

    label $mix.labmixddf -text "Select a ddf file : " -anchor e
grid  $mix.labmixddf -column 0 -row 2 -pady 0.5c -sticky w
    entry $mix.entmixddf -width 20 -textvariable DdfFileNameMix\
      -background white
grid  $mix.entmixddf -column 1 -row 2 -pady 0.5c -sticky we
    button $mix.browsemixddf -text "Browse ..." \
      -command "fileDialogDdfmix $mix $mix.entmixddf ddf"
grid  $mix.browsemixddf -column 2 -row 2 -pady 0.5c -sticky w

    label $mix.labmixsce -text "Select a sce file : " -anchor e
grid  $mix.labmixsce -column 0 -row 3 -pady 0.5c -sticky w
    entry $mix.entmixsce -width 20 -textvariable SceFileNameMix\
      -background white
grid  $mix.entmixsce -column 1 -row 3 -pady 0.5c -sticky we
    button $mix.browsemixsce -text "Browse ..." \
      -command "fileDialogScemix $mix $mix.entmixsce sce"
grid  $mix.browsemixsce -column 2 -row 3 -pady 0.5c -sticky w

proc fileDialogDdfmix {w ent operation} {
    #   Type names            Extension(s)      Mac File Type(s)
    #
    #---------------------------------------------------------
    set types {
      {"ddf files"            {.ddf}      }
      {"All files"            *}
    }
    if {$operation == "ddf"} {
      set file [tk_getOpenFile -filetypes $types -parent $w]
    } else {
```

```
      set file [tk_getOpenFile -filetypes $types -parent $w]
    }
    if [string compare $file ""] {
      $ent delete 0 end
      $ent insert 0 $file
      $ent xview end
    }
}

proc fileDialogScemix {w ent operation} {
    #   Type names            Extension(s)      Mac File Type(s)
    #
    #---------------------------------------------------------
    set types {
      {"Scenrio files"  {.sce}       }
      {"Module files"         {.asa}       }
      {"All files"            *}
    }
    if {$operation == "ddf"} {
      set file [tk_getOpenFile -filetypes $types -parent $w]
    } else {
      set file [tk_getOpenFile -filetypes $types -parent $w]
    }
    if [string compare $file ""] {
      $ent delete 0 end
      $ent insert 0 $file
      $ent xview end
    }
   }
 }

#////////////////////////////////////////////////////
#Cette fenetre est destinée à afficher les résultats
#de l'anlyse-----------------------------------------
#-------------------------------------------------
proc AffRst { } {
global APPLICAT win EDITEUR ddf app rst
#set rst .toprst
toplevel $rst
wm title $rst "$APPLICAT SESSION for REPORT"
wm minsize $rst 600 500
frame $rst.win -relief raised  -bd 2
pack $rst.win -fill both

#-------------------------------------------------------------
# Creation d'un menu
#-------------------------------------------------------------
menubutton $rst.win.file -text File -underline 0\
      -menu $rst.win.file.menu \
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*

menubutton $rst.win.options -text Options -underline 0\
      -menu $rst.win.options.menu \
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*

menubutton $rst.win.help -text Help -underline 0\
      -menu $rst.win.help.menu\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
```

```
pack $rst.win.file $rst.win.options $rst.win.help\
      -side left -fill x
menu $rst.win.file.menu
menu $rst.win.options.menu
menu $rst.win.help.menu
menu $rst.win.options.menu.analyse

$rst.win.file.menu add command -label "New..." \
      -command {NewWin_create;grab set $new_win}\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$rst.win.file.menu add command -label "Open..." \
      -command {set EDITEUR $rst ;set Fnp *.ddf;Fen_Open ;\
      open_dir "$env(HOME)"}\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$rst.win.file.menu add command -label "Close" \
      -command {destroy $rst}\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$rst.win.file.menu add separator
$rst.win.file.menu add command -label "Save" \
      -command "Save"\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$rst.win.file.menu add command -label "Save as..." \
      -command {set EDITEUR $rst ;set Fnp "*.ddf";Fen_Save ;\
      print_dir "$env(HOME)"}\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$rst.win.file.menu add separator
$rst.win.file.menu add command -label "Exit" \
      -command {exit} \
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$rst.win.options.menu add cascade -label "Start Analysis" \
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*\
      -menu $rst.win.options.menu.analyse
$rst.win.options.menu.analyse add command -label "Standard-mode
analysis..."\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*\
      -command {Fen_Start }
$rst.win.options.menu.analyse add command -label "Conversion-mode
analysis..."\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*\
      -command {Fen_Conv }
$rst.win.options.menu.analyse add command -label "Mixed-mode analysis..."\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*\
      -command {Fen_Mixed }
$rst.win.options.menu add separator
$rst.win.options.menu add command -label "Add Module..." \
      -command {Fen_Add ; mdl_dir "$env(HOME)"}\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-* -state disabled
$rst.win.options.menu add command -label "Delete Module" \
      -command {Delete Module} -state disabled\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$rst.win.options.menu add command -label "Check" \
      -command {Check} -state disabled\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$rst.win.options.menu add separator
$rst.win.options.menu add command -label "C-routines library" \
      -command {CroutinesLibrary}\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$rst.win.options.menu add command -label "Application Identfier..." \
      -command {Fen_AppId}\
```

```
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$rst.win.options.menu add command -label "Editor..." \
      -command { Fen_Edit }\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$rst.win.help.menu add command -label "Help on context"\
      -command {HelpOnContext} -state disabled \
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$rst.win.help.menu add command -label "Help on window"\
      -command {HelpOnWindow} -state disabled \
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$rst.win.help.menu add command -label "Help on help"\
      -command {HelpOnContext} -state disabled \
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*
$rst.win.help.menu add command -label "Help on version"\
      -command { Fen_Helpv }\
      -font *-*-medium-r-*-*-14-*-*-*-*-*-*-*

#Creation d'un editeur de text

text $rst.text -relief flat \
      -yscrollcommand "$rst.scrv set" -bg white\
      -font *-*-medium-*-*-*-14-*-*-*-*-*-*-*

focus $rst.text

# creation d'un scrollbar

scrollbar $rst.scrv  -command "$rst.text yview"
pack $rst.text -expand 1 -fill both -side left
pack $rst.scrv -side right -fill y
}

#////////////////////////////////////////////
#--------------------------------------------
#Boite de dialogue du menu add module
#Commentaires voir menu open
#--------------------------------------------
#--------------------------------------------
proc Fen_Add { } {
global mdl Fnp Chosf APPLICAT
set mdl .add
toplevel $mdl -bd 1c
set APPLICAT "ASAX"
wm title $mdl "$APPLICAT : Add Module..."
wm minsize $mdl 600 380
wm maxsize $mdl 600 400
wm geometry $mdl +150+100

trace variable Chosf w testvar

label $mdl.fnp -text "File name pattern"
grid $mdl.fnp -column 0 -row 0 -sticky w
entry $mdl.entfnp -textvariable Fnp \
      -background white -state disabled
set Fnp "*.asa"
grid $mdl.entfnp -column 0 -row 1 -sticky w
label $mdl.cdir -textvariable textvar
grid $mdl.cdir -column 0 -row 2 -sticky wn
label $mdl.files -text Files
```

```
grid $mdl.files -column 0 -row 2 -sticky ws
label $mdl.direct -text "Directories : "
grid $mdl.direct -column 2 -row 2 -sticky w
frame $mdl.frame1
grid $mdl.frame1 -column 0 -row 3 -sticky w
scrollbar $mdl.frame1.scr -command {$mdl.frame1.list yview}
grid $mdl.frame1.scr -column 1 -row 3 -sticky n,s
listbox $mdl.frame1.list -selectmode single -background white\
      -yscrollcommand {$mdl.frame1.scr set} -width 20 -height 5
grid $mdl.frame1.list -column 0 -row 3 -sticky w

frame $mdl.frame2
grid $mdl.frame2 -column 2 -row 3 -sticky w
scrollbar $mdl.frame2.scroll -command {$mdl.frame2.list yview}
grid $mdl.frame2.scroll -column 3 -row 3 -sticky n,s
listbox $mdl.frame2.list -selectmode single -background white\
      -yscrollcommand {$mdl.frame2.scroll set}  -width 20 -height 5
grid $mdl.frame2.list -column 2 -row 3 -sticky w

bind $mdl.frame2.list <Double-ButtonPress-1> {
    $mdl.entchosf delete 0 end
    set num [$mdl.frame2.list curselection]
    if {$num != "" } {
      mdl_dir [$mdl.frame2.list get $num]
set CurrDir [pwd]
    set NewCurrDir [split [string trim $CurrDir /] /]
set textvar "Current Dir : [lindex $NewCurrDir end]\n"
    }
break
}

bind $mdl.frame1.list <Double-ButtonPress-1> {
    set num [$mdl.frame1.list curselection]
    if {$num != "" } {
      FileName [$mdl.frame1.list get $num]
    }
break
}

label $mdl.chosf -text "Chosen File :"
grid $mdl.chosf -column 0 -row 4 -sticky w
entry $mdl.entchosf -textvariable Chosf -background white\
      -font *-*-bold-*-*-*-*-*-*-*-*-*-*-*

grid $mdl.entchosf -columnspan 2 -row 5 -sticky w,e
button $mdl.add -text Add -state disabled\
      -command {InsertMdl $Chosf; destroy $mdl}
grid $mdl.add -column 0 -row 6  -pady 5m -padx 5m
button $mdl.update -text Update -command { set Fnp "*.asa" ;\
      mdl_dir $env(HOME) ; $mdl.entchosf delete 0 end  ;\
      $mdl.add configure -state disabled }
grid $mdl.update -column 1 -row 6 -padx 5m
button $mdl.cancel -text Cancel -command { $mdl.entchosf delete 0 20 ; \
      destroy $mdl }
grid $mdl.cancel -column 2 -row 6 -padx 5m
button $mdl.help -text Help
grid $mdl.help -column 3 -row 6 -padx 5m

proc FileName { nomFichier } {
```

```
global mdl , Chosf
set Chosf $nomFichier
bind $mdl.entchosf $Chosf
}

proc testvar {nom element opr} {
global mdl
    if {$nom != ""} {
 $mdl.add configure -state active
   }
}

proc ChangeDir {dir} {
global mdl
bind $mdl.cdir $dir
}

bind $mdl.entfnp <Return> { bind $mdl.entfnp $Fnp ; mdl_dir "$env(HOME)";\
                            $mdl.entfnp configure -state disabled ; \
                        $mdl.entchosf delete 0 end }
}

proc mdl_dir { dir} {
    global mdl , Fnp , file , env , textvar
    $mdl.frame1.list delete 0 end
    $mdl.frame2.list delete 0 end
    bind $mdl.entfnp $Fnp
set CurrDir $dir
    set DIR [split [string trim $CurrDir /] /]
    set textvar "Current Dir : [lindex $DIR end]\n"
ChangeDirMdl textvar
    cd $dir
    $mdl.frame2.list insert end ".."
    foreach file [lsort [glob -nocomplain *] ] {
      if { [file isdirectory $file] } {
          $mdl.frame2.list insert end "$file/"
      } elseif { [file extension $file] == [string trim $Fnp *] } {
        $mdl.frame1.list insert end "$file"
      }
    }
}
proc ChangeDirMdl {dir} {
global mdl
bind $mdl.cdir $dir
}
#-------------------------------------------------------------
#Cette procedure permet d'inserer le fichier selectionne
#dans le scenario choisi
#-------------------------------------------------------------
proc InsertMdl {MdlFileName} {
global sce
    if {[regexp $MdlFileName [$sce.list get 0 end]] == 0} {
      $sce.list insert end $MdlFileName
     } else {Avertir $MdlFileName
            }
      }
#-------------------------------------------------------------------
#Cette procédure est appelée si le module à insérer
#existe déja dans le scénario.
```

```
#-------------------------------------------------------------
proc Avertir {fichier} {
      global Chosf APPLICAT
      toplevel .avert
      wm title .avert "$APPLICAT : Message..."
      wm geometry .avert +150+100
      message .avert.mesg -width 15c -justify left -relief flat -bd 2\
      -font -Adobe-Helvetica-Medium-R-Normal--*-180-*\
      -text "\n\nThe module named $fichier alredy exist in this scenario."\
      -justify center
      pack  .avert.mesg
      button .avert.btnOk -text Ok -command {destroy .avert}
      pack .avert.btnOk -ipadx 1c -pady 1c
      }
#/////////////////////////////////////////////////////////////////////
#---------------------------------------------------------------------
#Boite de dialogue du menu application identifer
#---------------------------------------------------------------------
#---------------------------------------------------------------------
proc Fen_AppId { } {
global app APPLICAT Name_APP ddf sce fen_mdl
set app .appid
toplevel $app -bd 5m
#set APPLICAT ASAX
wm title $app "$APPLICAT : Application identifier"
wm maxsize $app 550 200

trace variable Name_APP w testAPP

label $app.label -text "Application identifier : "
grid $app.label -column 1 -row 0 -pady 1c -sticky w
entry $app.entry -textvariable Name_APP -background white
grid $app.entry -column 2 -row 0
button $app.ok -text Ok -state disabled \
      -command {N_AppId $Name_APP}
grid $app.ok -column 0 -row 1 -pady 1c -padx 1c
button $app.reset -text Reset -command { $app.entry delete 0 end ;\
      $app.ok configure -state disabled ; set APPLICAT "ASAX";\
      N_AppId $APPLICAT}
grid $app.reset -column 1 -row 1
button $app.cancel -text Cancel -command { $app.entry delete 0 end ;
destroy $app }
grid $app.cancel -column 2 -row 1
button $app.help -text Help
grid $app.help -column 3 -row 1 -padx 1c
}
proc testAPP {nom element opr} {
global app
    if {$nom != ""} {
 $app.ok configure -state active
   }
 }

#/////////////////////////////////////////////////////////////
#-------------------------------------------------------------
#Boite de dialogue du menu editor
#-------------------------------------------------------------
#-------------------------------------------------------------
proc Fen_Edit { } {
```

```
global ed APPLICAT
set ed .edit
toplevel $ed -bd 5m
wm title $ed "$APPLICAT : Editor"
wm maxsize $ed 550 200
label $ed.label -text "Editor : "
grid $ed.label -column 1 -row 0 -pady 1c -sticky w
entry $ed.entry -text textvariable -background white
grid $ed.entry -column 2 -row 0
button $ed.ok -text Ok
grid $ed.ok -column 0 -row 1 -pady 1c -padx 1c -sticky w
button $ed.reset -text Reset -command { $ed.entry delete 0 end }
grid $ed.reset -column 1 -row 1  -padx 1c
button $ed.cancel -text Cancel -command {$ed.entry delete 0 end ; destroy
$ed}
grid $ed.cancel -column 2 -row 1 -padx 1c
button $ed.help -text Help
grid $ed.help -column 3 -row 1 -padx 1c
}
#////////////////////////////////////////////////////////////
#----------------------------------------------------------
#Boite  de dialogue du menu Help on version
#----------------------------------------------------------
#----------------------------------------------------------
proc Fen_Helpv { } {
global hp APPLICAT
set hp .help
toplevel $hp -bd 1c
wm title $hp "$APPLICAT : Help on version"
wm maxsize $hp 420 220
label $hp.label -text "SIEMENS\n --------\n \
      NIXDORF \n\n ASAX V 01 \n\n Copyright (c)\n \
      Siemens Nixdorf Informations systems AG 1993 \n \
      All Rights Reserved "\
      -pady 5m -padx 1c
button $hp.ok -text Ok -command { destroy $hp }
pack $hp.label $hp.ok
}
#////////////////////////////////////////////////////////////
#----------------------------------------------------------
#Cette procedure permet de configurer le nom de l'application
#----------------------------------------------------------
#----------------------------------------------------------
proc N_AppId {Name} {
global APPLICAT app win op sav ed hp ddf
set APPLICAT $Name
wm title $app "$APPLICAT : Application identifier"
wm title . "$APPLICAT SESSION"
wm title $ddf "$APPLICAT SESSION for DDF"
}
```